C 13.10:500-98

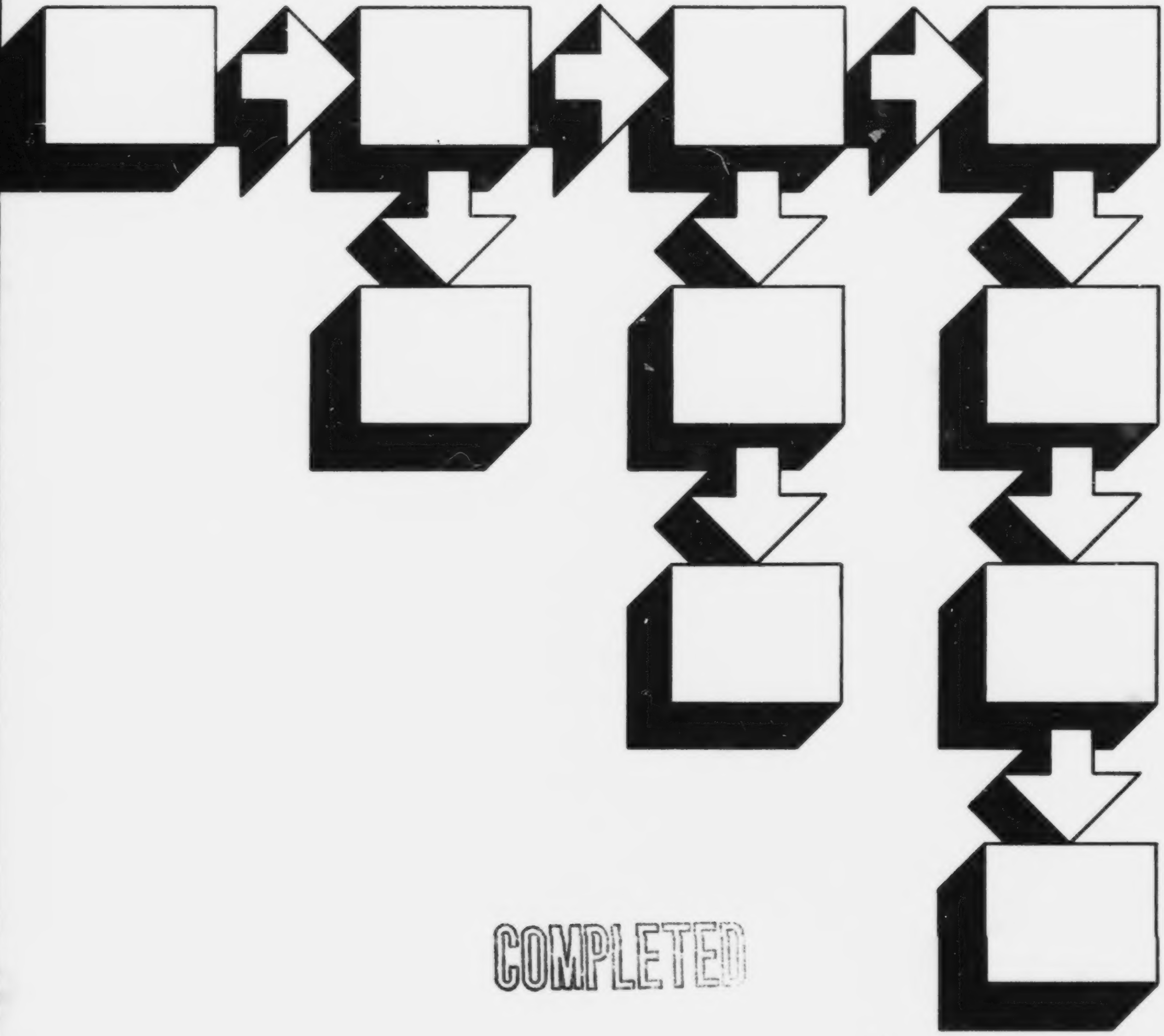
U.S. Department
of Commerce

National Bureau
of Standards

Computer Science and Technology

NBS Special Publication 500-98

Planning for Software Validation, Verification, and Testing



COMPLETED

S/N 003-003-02449-0 price $5.50

NATIONAL BUREAU OF STANDARDS

The National Bureau of Standards[1] was established by an act of Congress on March 3, 1901. The Bureau's overall goal is to strengthen and advance the Nation's science and technology and facilitate their effective application for public benefit. To this end, the Bureau conducts research and provides: (1) a basis for the Nation's physical measurement system, (2) scientific and technological services for industry and government, (3) a technical basis for equity in trade, and (4) technical services to promote public safety. The Bureau's technical work is performed by the National Measurement Laboratory, the National Engineering Laboratory, and the Institute for Computer Sciences and Technology.

THE NATIONAL MEASUREMENT LABORATORY provides the national system of physical and chemical and materials measurement; coordinates the system with measurement systems of other nations and furnishes essential services leading to accurate and uniform physical and chemical measurement throughout the Nation's scientific community, industry, and commerce; conducts materials research leading to improved methods of measurement, standards, and data on the properties of materials needed by industry, commerce, educational institutions, and Government; provides advisory and research services to other Government agencies; develops, produces, and distributes Standard Reference Materials; and provides calibration services. The Laboratory consists of the following centers:

Absolute Physical Quantities[2] — Radiation Research — Chemical Physics — Analytical Chemistry — Materials Science

THE NATIONAL ENGINEERING LABORATORY provides technology and technical services to the public and private sectors to address national needs and to solve national problems; conducts research in engineering and applied science in support of these efforts; builds and maintains competence in the necessary disciplines required to carry out this research and technical service; develops engineering data and measurement capabilities; provides engineering measurement traceability services; develops test methods and proposes engineering standards and code changes; develops and proposes new engineering practices; and develops and improves mechanisms to transfer results of its research to the ultimate user. The Laboratory consists of the following centers:

Applied Mathematics — Electronics and Electrical Engineering[2] — Manufacturing Engineering — Building Technology — Fire Research — Chemical Engineering[2]

THE INSTITUTE FOR COMPUTER SCIENCES AND TECHNOLOGY conducts research and provides scientific and technical services to aid Federal agencies in the selection, acquisition, application, and use of computer technology to improve effectiveness and economy in Government operations in accordance with Public Law 89-306 (40 U.S.C. 759), relevant Executive Orders, and other directives; carries out this mission by managing the Federal Information Processing Standards Program, developing Federal ADP standards guidelines, and managing Federal participation in ADP voluntary standardization activities; provides scientific and technological advisory services and assistance to Federal agencies; and provides the technical foundation for computer-related policies of the Federal Government. The Institute consists of the following centers:

Programming Science and Technology — Computer Systems Engineering.

[1] Headquarters and Laboratories at Gaithersburg, MD, unless otherwise noted; mailing address Washington, DC 20234.

[2] Some divisions within the center are located at Boulder, CO 80303.

Computer Science and Technology

NBS Special Publication 500-98

Planning for Software Validation, Verification, and Testing

Patricia B. Powell, Editor

Center for Programming Science and Technology
Institute for Computer Sciences and Technology
National Bureau of Standards
Washington, DC 20234

U.S. DEPARTMENT OF COMMERCE
Malcolm Baldrige, Secretary

National Bureau of Standards
Ernest Ambler, Director

Issued November 1982

Reports on Computer Science and Technology

The National Bureau of Standards has a special responsibility within the Federal Government for computer science and technology activities. The programs of the NBS Institute for Computer Sciences and Technology are designed to provide ADP standards, guidelines, and technical advisory services to improve the effectiveness of computer utilization in the Federal sector, and to perform appropriate research and development efforts as foundation for such activities and programs. This publication series will report these NBS efforts to the Federal computer community as well as to interested specialists in the academic and private sectors. Those wishing to receive notices of publications in this series should complete and return the form at the end of this publication.

National Bureau of Standards Special Publication 500-98
Natl. Bur. Stand. (U.S.), Spec. Publ. 500-98, 89 pages (Nov. 1982)
CODEN: XNBSAV

Library of Congress Catalog Card Number: 82-600644

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON 1982

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, DC 20402
Price $5.50
(Add 25 percent for other than U.S. mailing)

CONTENTS

Page

Chapter 1 INTRODUCTION 1

1.1 AUDIENCE 2

1.2 PHILOSOPHY 2

1.3 USER OF THIS DOCUMENT 2

Chapter 2 AN OVERVIEW OF SOFTWARE DEVELOPMENT 4

2.1 SOFTWARE VALIDATION, VERIFICATION, AND TESTING - EVALUATION AND REVIEW THROUGHOUT THE PROBLEM SOLVING PROCESS 4

2.2 A PROBLEM SOLVING MODEL FOR SOFTWARE DEVELOPMENT 6

- 2.2.1 Initiation Phase 7
- 2.2.2 Development Phase
  - 2.2.2.1 Requirements Subphase 8
  - 2.2.2.2 Preliminary Design Subphase 9
  - 2.2.2.3 Detailed Design Subphase 10
  - 2.2.2.4 Programming and Testing Subphase 11
  - 2.2.2.5 Installation Subphase 12
- 2.2.3 Operation and Maintenance Phase 13

Chapter 3 A FRAMEWORK FOR INTEGRATED VALIDATION, VERIFICATION, AND TESTING 15

3.1 STATIC ANALYSIS 15

3.2 DYNAMIC ANALYSIS 16

Page

3.3 FORMAL ANALYSIS 19

3.4 AN INTEGRATED APPROACH TO VALIDATION, VERIFICATION, AND TESTING 20

3.5 REQUIREMENTS VALIDATION, VERIFICATION, AND TESTING 21

3.6 DESIGN VALIDATION, VERIFICATION, AND TESTING 23

3.7 CODE VALIDATION, VERIFICATION, AND TESTING 25

3.8 SUMMARY 27

Chapter 4 VALIDATION, VERIFICATION, AND TEST PLANNING 28

4.1 VALIDATION, VERIFICATION, AND TEST PLANNING IN THE TOTAL PROJECT CONTEXT 28

4.2 STEP I: IDENTIFYING V,V&T GOALS 32

4.3 STEP II: DETERMINE INFLUENCES ON V,V&T ACTIVITIES 35

4.4 STEP III: SELECT V,V&T TECHNIQUES AND TOOLS 37

4.5 STEP IV: DEVELOP A DETAILED V,V&T PLAN 40

Chapter 5 EXAMPLE APPLICATIONS OF VALIDATION, VERIFICATION, AND TESTING TECHNOLOGY 43

5.1 OVERVIEW OF EXAMPLES 43

5.2 EXAMPLE 1: SOFTWARE DEVELOPMENT USING MANUAL V,V&T TECHNIQUES 46
  5.2.1 Requirements Subphase Activity Descriptions
    5.2.1.1 Initial Requirements Review 47
    5.2.1.2 Requirements Analysis 47
    5.2.1.3 V,V&T Planning 51
    5.2.1.4 Initial Test Case Generation 51
  5.2.2 Preliminary Design Subphase Activity Descriptions
    5.2.2.1 Refinement of Graphical Representation 52
    5.2.2.2 Specify Information Design 52
    5.2.2.3 Design Program Architecture 53
    5.2.2.4 Design Basic Control Flow 53
    5.2.2.5 Test Case Generation 54
    5.2.2.6 Preliminary Design Review 54

Page

5.2.3 Detailed Design Subphase Activity Descriptions
5.2.3.1 Detailed Database Design 54
5.2.3.2 Detailed Module Design 55
5.2.3.3 Test Case Generation 56
5.2.3.4 Critical Design Review 56
5.2.4 Programming Subphase Activity Description
5.2.4.1 Code Development 56
5.2.4.2 Module Testing 57
5.2.4.3 Function Testing 58
5.2.4.4 Acceptance Testing 58

5.3 EXAMPLE 2: SOFTWARE DEVELOPMENT USING A MINIMAL AUTOMATED V,V&T TOOL SET 58
5.3.1 Requirements Subphase Activity Description 59
5.3.2 Preliminary Design Subphase Activity Descriptions
5.3.2.1 Specify Information Design 59
5.3.2.2 Design Basic Control Flow 60
5.3.3 Detailed Design Subphase Activity Description 60
5.3.4 Programming Subphase Activity Descriptions
5.3.4.1 Code Development 61
5.3.4.2 Module Testing 61
5.3.4.3 Function Testing 62

5.4 EXAMPLE 3: SOFTWARE DEVELOPMENT USING A FULLY AUTOMATED V,V&T TOOL SET 63
5.4.1 Requirements Subphase Activity Description 63
5.4.2 Preliminary Design Subphase Activity Description 63
5.4.3 Detailed Design Subphase Activity Description 64
5.4.4 Programming Subphase Activity Descriptions
5.4.4.1 Code Development 65
5.4.4.2 Module Testing 66

5.5 EXAMPLE 4: SOFTWARE MAINTENANCE 67
5.5.1 Problem Reporting Activity Descriptions
5.5.1.1 Problem Analysis 68
5.5.1.2 Problem Correction 69
5.5.2 Change Request Activity Descriptions
5.5.2.1 Requirements Analysis 70
5.5.2.2 Redesign and Coding 70

Chapter 6 SUMMARY 71

Glossary 72

Bibliography 75

Index

LIST OF FIGURES

Page

Figure 1.1-1 Reading Scenarios 3
Figure 2.1-1 Three Categories of V,V&T Activities 5
Figure 2.2-1 Summary of V,V&T Activities 6
Figure 3.1-1 General Form of Static Analysis 15
Figure 3.1-2 Module Interface Consistency Checks 16
Figure 3.2-1 General Form of Dynamic Analysis 17
Figure 3.3-1 General Form of a Formal Functional Analysis 20
Figure 3.4-1 General V,V&T Integration Strategy 21
Figure 3.5-1 Integrated Approach to Requirements V,V&T 22
Figure 3.6-1 Integrated Approach to Design V,V&T 23
Figure 3.7-1 Integrated Approach to Code V,V&T 26
Figure 4.1-1 Software Project Planning 29
Figure 4.1-2 Evolution of Project Plan 29
Figure 4.1-3 Preparation of Plans 30
Figure 4.1-4 Abbreviated Outline of a Project V,V&T Plan 31
Figure 4.2-1 Goals and Measurement Criteria 33
Figure 4.4-1 V,V&T Technique and Tool Selection Worksheet 38
Figure 4.4-2 Completed V,V&T Technique and Tool Selection Worksheet 39
Figure 5.1-1 Overview of Examples 44
Figure 5.1-2 Example Software Development Lifecycle 44
Figure 5.2-1 Informal Prose Requirements 48
Figure 5.2-2 Requirements Graphical Representation 50
Figure 5.2-3 Sample Database Schema Showing Client-Claims Relation 54
Figure 5.2-4 Sample CLAIMS Record Description 55
Figure 5.2-5 Sample Portion of Code Inspection Checklist 57
Figure 5.3-1 PSL Specifications for Accounts Payable 59
Figure 5.3-2 Preliminary Design PDL of "Issue Policy Notices" Function 60
Figure 5.3-3 Detailed Design PDL of "Issue Policy Notices" Function 61
Figure 5.4-1 Detailed PDL with Assertions 65
Figure 5.4-2 Find-policy Subroutine and Corresponding Assertion Violation Message 67

LIST OF TABLES

Page

Table 5.2-1 Example 1 Summary - Software Development Using Manual V,V&T Techniques 46
Table 5.3-1 Example 2 Summary - Software Development Using a Minimally Automated V,V&T Tool Set 58
Table 5.4-1 Example 3 Summary - Software Development Using a Fully Automated V,V&T Tool Set 63
Table 5.5-1 Example 4 Summary - Software Maintenance 68

Abstract

Today, providing computer software involves greater cost and risk than providing computer equipment. One major reason is hardware is mass-produced by proven technology, while software is still produced primarily by the craft of individual computer programmers. The document is for those who direct and those who implement computer projects; it explains the selection and use of validation, verification, and testing (V,V&T) tools and techniques for software development. A primary benefit of practicing V,V&T is increasing confidence in the quality of the software. The document explains how to develop a plan to meet specific software V,V&T goals.

Key words: automated software tools; software lifecycle; software testing; software verification; test coverage; test data generation; validation.

ACKNOWLEDGMENTS

This report was funded by the National Bureau of Standards' Institute for Computer Sciences and Technology under U.S. Department of Commerce Contract NB79SBCA0102. The contributors to the report, as submitted by Boeing Computer Services were Randy L. Merilatt, Mark K. Smith, and Leonard L. Tripp, assisted by Allen R. Bennett, John R. Brown, Susan C. Chew, Linda S. Hammond, William E. Howden, Leon J. Osterweil, and Richard N. Taylor. Consultation was provided by Leon G. Stucki.

PREFACE

The following document was originally included as part of a report titled "Computer Software Validation and Verification: A General Guideline". The chapter on techniques and tools was extracted and published as a reference manual [POWE]; it is a companion to this document. This document, being prepared under contract to the Institute for Computer Sciences and Technology, is in the public domain and is, therefore, not subject to copyright. Acknowledgment and thanks are appropriate for the following reviewers who donated their time and energy to critiquing the document:

John B. Bowen
Martha A. Branstad
Marianne Freedman
Carolyn Gannon
Herbert Hecht
Raymond C. Houghton, Jr.
Melba Hye-Knudsen
Denise Kreuss
Sukahamay Kundu
Frank LaMonica
David Markham
Ralph Neeper
Jack L. Nelson
Albrecht Newmann
David E. Nichols
Joanne Pasco
Sam Redwine
Ard Robertson
Harlan Seyfer
Jim Skiles
Al Sorkowitz
Susan Voight
John Walter
Alice Wong
Natalie Yopconka
Saul Zaveler

Comments pertaining to the technical content should be directed to:

Systems and Software Technology Division
Room B266 Bldg. 225
National Bureau of Standards
Washington, D.C. 20234

CHAPTER ONE

INTRODUCTION

The Institute for Computer Sciences and Technology (ICST) carries out the following responsibilities under P.L. 89-306 (Brooks Act) to improve the Federal Government's management and use of ADP:

- develops Federal automatic data processing standards;
- provides agencies with scientific and technological advisory services relating to ADP;
- undertakes necessary research in computer sciences and technology.

In partial fulfillment of Brooks Act responsibilities, ICST issues Special Publications (S.P.). This document describes an approach to validation, verification, and testing of computer software that pervades the entire development and maintenance process. The document consists of four additional chapters which can be grouped into two sections, described below.

Software development is an exercise in problem solving. The solution is embodied in the final product, the computer software. This product consists of the programs (computer instructions) and the manuals describing the software and its use. Executing the program with data on a computer provides the solution. During the development of the target software, there are several intermediate products produced by the project requestor and the developers. The methodology to enhance the overall correctness of the final product by working with the intermediate products is the subject of this document.

In problem solving, a key activity is to determine that the solution is correct. Validation, Verification, and Testing (V,V&T) is a process composed of the set of procedures, activities, techniques and tools used to ensure that a software product does solve the intended problem.

The chapters in this report are grouped into two principal sections: V,V&T Background and V,V&T Planning Guide.

V,V&T Background consists of two chapters. Chapter 2 describes a phased approach to software development and the fundamental concepts of V,V&T. Chapter 3 describes three classes of V,V&T techniques and tools, and a scheme for integrating them.

V,V&T Planning Guide consists of two chapters. Chapter 4 explains how the goals for V,V&T can be identified and how to develop a project V,V&T plan. Chapter 5 discusses the application of V,V&T principles through examples.

1.1 AUDIENCE

This document is directed toward those (managers, customers, etc.) who influence how software development is done and to those (programmers, analysts, etc.) who do development. The document assists a project manager working with the customer in establishing V,V&T goals. It further assists project managers in developing a plan for achieving the goals. Finally it guides in the judicious selection of the appropriate set of V,V&T practices, techniques, and tools.

To the software engineer who is responsible for performing the various V,V&T functions, this document and its companion, the Technique and Tool Reference Guide[POWE], provide guidance to the actual use of the selected techniques and tools. For each technique or tool there is information including functions, inputs, outputs, resources required for use, and sources for more detailed information. In addition to aiding in the application of individual techniques and tools, information on their integrated use is presented.

The document is also a resource to another group within the ADP environment in that it addresses certain needs of the ADP policy maker. It presents certain fundamental concepts, elements of a general V,V&T approach, and many of the specifics necessary to implement the approach. It provides information that may be helpful in forming a basis for policies relating to V,V&T practice. Second, the document contains information relevant to the formulation and implementation of the software V,V&T functions in a typical ADP environment.

1.2 PHILOSOPHY

This document provides assistance in all aspects of V,V&T. It is not a total "project cookbook," wherein all the techniques and tools must be used on every project. Projects vary in size, scope, complexity, and other characteristics that influence the specific V,V&T approach. Each project must be evaluated to determine how V,V&T might be applied.

The use of V,V&T does not, of itself, guarantee success. It requires judgement, training, and experience. Like other methodologies, the application of V,V&T may be good or bad; but, it should never be blind.

1.3 USER OF THIS DOCUMENT

For the reader who is encountering this document for the first time, reading Chapters 1 through 3 is recommended. Reading of subsequent chapters should be guided by intended use of the information. Suggested reading scenarios for three types of readers are presented in Figure 1.1-1. "Validation, Verification, and Testing - Technique and Tool Guide"[POWE] is a companion to the general guidance in this document.

	V,V&T BACKGROUND			V,V&T PLANNING GUIDE	
Chapters	1	2	3	4	5
	- POLICY MAKER -			- - - PLANNER - - -	
				- - - SELECTOR - - -	

Figure 1.1-1 Reading Scenarios

CHAPTER TWO

AN OVERVIEW OF SOFTWARE DEVELOPMENT

Systematic approaches to problem solving, including software development, proceed through a sequence of steps or phases with probable looping back to previous phases. This type of approach yields several important benefits.

First, because the problem solving process is subdivided into separate phases, the problem solver is eased into a gradual process. The problem solver is able to approach separately the problem, the solution, and its implementation. The problem solver need not deliberate on all three at once. This allows larger, more complex problems to be resolved successfully.

Second, a phased process provides intermediate monitoring and control points. The existence of phases and intermediate products increases the visibility of the process. This encourages the involvement of the group for whom the problem is being solved and increases their control of the problem solving.

Third, the existence of a series of intermediate specifications, i.e. requirements, design, and code, of the solution facilitates early and continual evaluation of the solution as it moves toward implementation. In software development this process of continual review and evaluation is validation, verification, and testing (V,V&T).

This chapter discusses software development as a problem solving activity. It introduces the concepts of software V,V&T and a phased approach to software development. Specifically, it describes:

Software Validation, Verification, and Testing - Evaluation and Review Throughout the Problem Solving Process - The process of obtaining increasing levels of confidence in a solution through a series of checkpoints and reviews is discussed as it applies to software development and the analogies previously presented.

A Problem Solving Model for Software Development - A specific series of problem solving phases for software development is described, including the V,V&T activities associated with each.

2.1 SOFTWARE VALIDATION, VERIFICATION, AND TESTING - EVALUATION AND REVIEW THROUGHOUT THE PROBLEM SOLVING PROCESS

Validation, Verification, and Testing (V,V&T), in general terms, is a process of review, analysis, and testing employed throughout the software development lifecycle. It is a methodology which helps ensure the production of quality software. Validation determines the correctness of the end product, e.g. code, with respect to the software requirements, i.e. does the output conform with what was required? Verification is performed at each phase and between each phase of the development lifecycle. It determines that each phase and subphase product is correct, complete, and consistent with itself and with its predecessor product. Testing, either automated or manual, examines program behavior by executing the program on sample data sets, e.g. passing data, automatically or manually, through a design to determine the correctness of the program is testing the design.

V,V&T is commonly used as a single expression; this is not to infer that the methodology is an all or nothing process. Project constraints, such as criticality, error tolerance or budget, should determine how much validation, verification, or testing is applied to that project. This document uses the general term, V,V&T, referring to the total methodology; it is understood that the reader will extract those portions of V,V&T which are feasible and applicable to the specific situation. V,V&T may be performed by an independent V,V&T group (IV&V), by the person(s) producing the product or a combination of both. Again, the decision as to who performs the V,V&T is project dependent. The objective of V,V&T is to ensure the correctness, completeness, and consistency of the final product.

Software V,V&T focuses on the prevention and the detection of errors, i.e. deviation from intent. This is accomplished through the use of both manual and automated techniques. Errors include deficiencies such as unsatisfied requirements, or, the converse, the inclusion of extraneous functions. An error may be in the coding of the software, a specification of the software, (e.g., a design specification) or the documentation (e.g., a user's manual). The error might be related to the functional correctness or another property, such as performance, or a more subjective attribute such as product form.

To describe the role of V,V&T in error detection and prevention, three categories of V,V&T activities are described. These are illustrated in Figure 2.1-1.

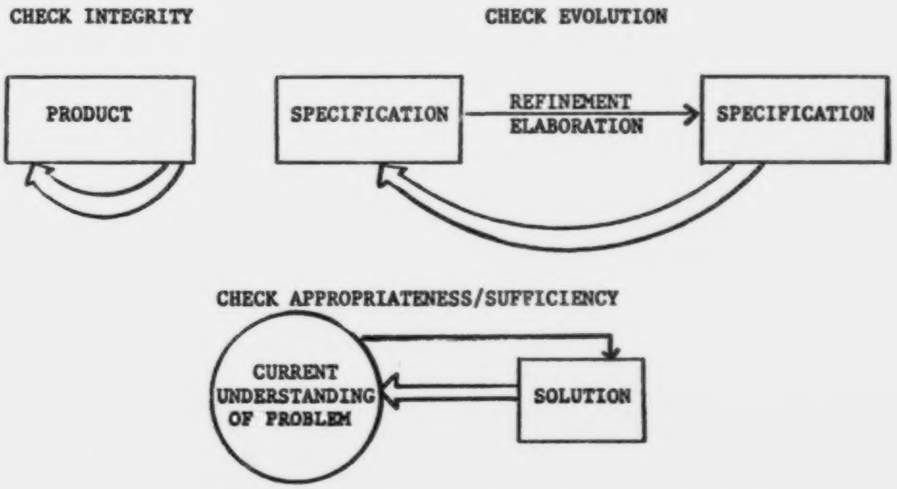


Figure 2.1-1 Three Categories of V,V&T Activities

The objective of integrity checking, the first category of V,V&T activities, is to verify the integrity of the products at each phase of development. Each product is analyzed for internal consistency and completeness. For example, a requirements specification can be analyzed to detect inconsistent or contradictory requirements such as the specification of an output report that requires data which is unavailable.

The objective of evolution checking, the second category, is to ensure the completeness and consistency between levels of specification, where the second is a refinement or elaboration of the first.

The objective of appropriateness/sufficiency checking, the third category, is to compare this evolving solution against the problem as currently understood to ensure that it is a necessary and sufficient solution.

2.2 A PROBLEM SOLVING MODEL FOR SOFTWARE DEVELOPMENT

DEVELOPMENT PHASE

Requirements Definition and Analysis Subphase

- Development of the Project V,V&T Plan
- Generating Requirements Based Test Cases
- Review and Analysis of the Requirements

Preliminary Design Subphase

- V,V&T Planning
- Generating Design Based Test Scenarios
- Review and Analyze the Preliminary Design

Detailed Design Subphase

- Generation of Design Based Functional Test Data
- Review and Analysis of the Detailed Specification

Programming and Testing Subphase

- Complete the Test Case Specification
- Review, Analysis, and Testing of the Program

Installation Subphase

- System Acceptance

OPERATION AND MAINTENANCE PHASE

- Software Evaluation
- Software Modification Evaluation
- Regression Testing

Figure 2.2-1 Summary of V,V&T Activities

The following discussion describes each phase/subphase by presenting the nature and purpose of each, the products produced, and the V,V&T activities which incrementally build confidence in the software product as it evolves. Figure 2.2-1 summarizes the V,V&T activities.

2.2.1 INITIATION PHASE

Description: The initiation phase begins with the recognition of a problem and concludes with a decision of whether or not to implement a software solution. The recognition of a problem may be sudden or gradual; the problem may be generally recognized or only perceived by a small group. The main activity of the initiation phase is a joint exploration of the problem by the group having the problem and the group responsible for solving it. The decision to pursue a solution must be based upon a clear understanding of the problem, a preliminary investigation of alternative solutions, and a comparison of the expected benefits versus the cost (design, construction, and operation) of the solution.

Products: The primary result is a decision about the continuation of the project. To support this decision, there are often three products. These are:

- The project request or proposal: defines the problem to be solved and the scope and objectives of the proposed solution.
- The feasibility study: states the assumptions being made, defines alternative solutions and assesses technical and operational feasibility.
- The cost/benefit analysis: estimates and compares the costs (construction and operation) and the expected benefits (both quantifiable and intangible).

V,V&T Activities: The three products are reviewed by the problem posers, the problem solvers, and the decision makers (e.g., higher level management in control of required resources). The following questions are generally answered in this review.

- Has the scope, cost, impact, and urgency of the problem been adequately defined?
- Has a technically feasible and operationally viable solution been identified?
- Have alternative solutions been identified?
- Have alternatives been adequately studied?
- Have the costs of the solutions been analyzed and compared against the expected benefits? What are the probabilities of achieving these benefits? What are the risks involved?

2.2.2 DEVELOPMENT PHASE

As an example, this phase is divided into five subphases for the purpose of illustration. These are each separately described.

2.2.2.1 REQUIREMENTS DEFINITION AND ANALYSIS SUBPHASE

Description: The goal of the requirements subphase is to put the problem into a rigorous form upon which a solution can be based i.e., a statement of the requirements which a software solution must satisfy. Requirements identification is iterative, involving the problem posers and the problem solvers. Requirements may be modified in later subphases as a better understanding of the problem is gained. These modifications are documented, creating a traceable record of the progress and evolution of the final product. Also during this subphase, two planning activities are performed. First, project plans, budgets, and schedules for the develoment phase and each subphase are developed. In addition, the V,V&T goals are identified and a plan for achieving these goals is developed.

Products: This subphase results in the preparation of four products. The first three are completed, while the fourth is finished during subsequent phases.

- o The software requirements document: Specifies what the system must do. This includes the requisite information flows and processing functions. Acceptance criteria for deciding that the requirements are satisfied as specified is an important part of this product.

- o The project plan: Specifies a strategy for managing the software development. It defines the goals and activities for all phases and subphases. It includes: resource estimates over time and intermediate milestones including management and technical reviews. It defines methods for design, documentation, problem reporting, and change control. It also specifies supporting techniques and tools.

- o The Validation, Verification, and Testing Plan: Specifies goals of the V,V&T activities including software testing. It is the design of a project specific V,V&T process and identifies techniques and tools to assist in achieving the goals. It specifies plans (schedules, budgets, responsibilities, etc.) for performing the V,V&T activities.

- o The Software Test Case Specification: Describes the scenarios and cases for testing each requirement. The acceptance criteria are used to develop test cases. Expected results for each test case are included.

V,V&T Activities: There are three basic thrusts to these activities. These are:

- o Development of the Project V,V&T Plan: Goals for V,V&T activities are determined; a V,V&T process is designed; techniques and tools are chosen; schedules and budgets are established.

- o Generating Requirements Based Test Cases: These form a basic set of test cases. They help clarify and determine the measurability of the software requirements and form a basis for acceptance testing.

- o Review and Analysis of the Requirements: The goal is to ensure that the requirements identified will result in a feasible and a usable solution to the entire problem. They are reviewed for clarity, completeness, correctness, consistency, testability, and traceability to the problem statement.

2.2.2.2 PRELIMINARY DESIGN SUBPHASE

Description: During preliminary design, the problem solvers (the software developers) assisted by the problem posers (the customer/user) formulate and analyze alternative solutions. This may reveal flaws in the requirements and result in its modification. This iteration continues until all issues have been resolved.

This subphase results in a high level specification of the solution. The solution is conceptual in nature, defining information aggregates, information flows, and logical processing steps. It will describe all the major interfaces, and their inputs and outputs. Implementation details, e.g., actual programs and physical data structures are generally not addressed.

Project plans (schedules, budgets, deliverables, etc.) are reviewed and revised as appropriate for the scope and complexity of the solution formulated.

Products: There is one new product of this subphase - the preliminary design specification. In addition, each of the four products of the requirements subphase may undergo revision or be supplemented with new data.

- o The Preliminary Design Specification: Documents the high level solution developed during this phase. This may be packaged in two separate documents, i.e., a system/subsystem specification and a data/database specification.

- o A Revised Requirements Specification: Design activities may reveal inconsistent, infeasible, or ambiguous requirements and result in the revision of their specification.

- o An Updated Project Plan: Upon the completion of the preliminary design, the scope and complexity of the solution are well understood. As a result, the project plan (schedules, budgets, deliverables, etc.) is made more accurate and realistic.

- o An Updated V,V&T Plan: This plan may warrant revision based upon new or revised requirements.

- o Software Test Case Specification: Additional test scenarios and test cases are developed to exercise and test aspects of the design.

V,V&T Activities: There are three areas of V,V&T activity:

- V,V&T Planning: The V,V&T plan is reviewed and revised as deemed necessary.
- Generating Design Based Test Scenarios: Complementing and expanding the requirements based test data generated focusing on the logical functions performed.
- Review and Analyze the Preliminary Design: To assure internal consistency, completeness, correctness and clarity; to verify that the design is linked to and, when implemented, will satisfy the requirements.

2.2.2.3 DETAILED DESIGN SUBPHASE

Description: The purpose of this subphase is to refine, resolve deficiencies, define additional details, and package the logical solution created in the previous subphase. Implementation details are addressed. Ambiguities are removed from the design specification. The detailed design specification describes the physical solution (algorithms and data structures) which is an elaboration of the logical solution specified in the preliminary design. The result is a solution specification that can be implemented in code with little or no need for additional analysis.

The detailed design team may discover processing operations that are impractical or impossible to implement, necessitating modification of the preliminary design and possibly the requirements as well.

The user and participants in the review are consulted in making major design decisions.

Products: There are two new products of this subphase and additional information added to an existing one:

- The Detailed Design Specification: A fully detailed description of the software (algorithms and data) to be coded in the following subphase.
- Software Test Case Specification: This is now a substantially complete description of test data and the expected results.
- Problem Reports: Formal statements of observed problems. This may necessitate going back to a previous subphase.

V,V&T Activities: The two V,V&T activities of this phase are:

- Generation of Design Based Functional Test Data: Formulated test data based on the physical structure of the system.
- Review and Analysis of the Detailed Specification: To assure internal

consistency, completeness, correctness, and clarity; to verify that the detailed design is linked to and is a correct refinement of the preliminary design; to validate that the design when implemented will satisfy the requirements.

2.2.2.4 PROGRAMMING AND TESTING SUBPHASE

Description: This subphase results in a program which is ready for installation. Programming is the process of implementing the detailed design specification into code. Only minor, if any, design issues are resolved during this subphase.

Completed code undergoes testing as described in the V,V&T plan. Generally, three types of testing are performed: unit, integration, and system. While unit testing is done by the programmer, the person(s) responsible for integration and system testing is project specific.

Unit testing checks for typographical, syntax, and logic errors. Each of the modules of code are checked individually by the programmers who wrote them to ensure that each correctly implements its design and satisfies the specified requirements.

Integration testing focuses on checking the intermodule communication links, and testing aggregate functions formed by groups of modules.

System testing examines the operation of the system as an entity. This type of testing ensures that the software requirements have been satisfied both singly and in combination.

The final activity of this subphase is planning the installation of the software.

Products: There are six new outputs produced during this subphase and one other which is completed.

- o Software Test Case Specification: Final revisions and additions to the test data are made.

- o Program Code: Fully documented and tested code, which is ready for installation.

- o Test Results and Test Evaluation Reports: The documentation of the comparison of actual and expected results.

- o User Documentation: Manuals describing the input and report formats, user commands, error messages, and instructions for operation by the user.

- o Maintenance Manual: Documentation to maintain the system.

- o Installation Plan: Specifies the approach to and details of the installation of the software.

- o Problem Reports: Formal statements of observed problems. This may necessitate going back to a previous subphase.

V,V&T Activities: The V,V&T activity of this phase focuses upon the program produced.

- o Complete the Test Case Specification: Final additions and modifications necessary due to design changes made during coding.
- o Review, analysis and testing of the program: Includes checking for adherence to coding standards, manual/automated analysis of the program, and the execution of the program on test data ensuring that it meets the acceptance criteria.

2.2.2.5 INSTALLATION SUBPHASE

Description: The result of this subphase is a system incorporating the developed programs, other software components, the hardware, and production data. The activities of the subphase are guided by the installation plan developed. The first task is to integrate the system components. Integration consists of installing hardware, installing the program(s) onto the computer, reformatting/creating the data base(s), and verifying that all components have been included. Modification to program code may be necessary to obtain compatibility between hardware and software, or between different software modules.

The next task is to test the system. The test data from earlier subphases is enhanced and used here. The result is a system qualified and accepted for production use.

The third task is the start of system operation. The strategies for this include immediate cutover, phased cutovers, or parallel operation. This task also includes operator and user training.

Products: The new product of this subphase is the Installation Report, but previously completed products may be updated to incorporate findings of the installation activity.

- o Installation Report: Describes the results of the installation activities, including data conversion, installation testing/results, and software/system problems and modifications necessary.

V,V&T Activities: The primary V,V&T activity centers on acceptance of the system by the customer. This could be a simple statement signed by a customer representative. This marks the end of the development phase.

2.2.3 OPERATION AND MAINTENANCE PHASE

Description: The final phase involves the actual use of the software and monitoring its operation to ensure that it succeeds in solving the user's problem.

Most often, some need for modifying the software arises during this phase. The maintenance process involves determining the reason for each modification. The cause could be an error made in the original development, the recognition of a new requirement, or the desire for a design modification to improve performance, usability, etc. Once the cause is determined, the software (code and documentation) is 'redeveloped' from that point. For example, the redevelopment due to a change in requirements will result in modifications to the requirements specification, the design, the code, and user and operation manuals.

Problem reporting, change requests, and other change control mechanisms are used to facilitate the systematic correction and evolution of the software. In addition, performance measurement and evaluation activities are performed to ensure that the system continues to meet its requirements in the context of a changing system environment.

Products: To track and manage the evolution of the software in this post development phase, several new outputs are produced:

- Problem reports: Formal statements of observed problems. The analyses of these may result in software change requests.
- Change requests: Requests for specific modification to the software. These could be generated due to an error (i.e., problem report) or a modification of the requirements or design.
- Revision to initial development products: As a result of change requests any one or all of the products of the initiation and development phases may require revision.

V,V&T Activities: The V,V&T activities of this phase can be separated into two categories: the monitoring and evaluation of the system, and the problem correction activities. The first is an on-going activity. The second is driven by the problem reports and change requests.

- Software evaluation: Activities aimed at assessing the operation of the software and assuring continued satisfaction of user requirements.
- Software modification evaluation: As needs for change are discovered and requests for modifications are made, the requested modifications are evaluated in the same manner as the original software is evaluated. After modifications are made, the changes are reviewed and tested to ensure that the expected changes in the software is achieved.
- Regression testing: Rerunning test cases which a program has previously executed correctly in order to detect errors created during software

correction/modification activities.

CHAPTER THREE

A FRAMEWORK FOR INTEGRATED VALIDATION, VERIFICATION, AND TESTING

To do effective V,V&T, the development team needs to select a well-matched, compatible set of techniques and tools. The selection is based on the V,V&T goals of the project. These goals take into account the characteristics of problem and the constraints on the solution. From the user's perspective, the techniques and tools must ensure functional, efficient, reliable and low maintenance software. The developer is concerned with these issues as well as the integrity and compostion of the software. This blend of concerns is the guiding force in selecting V,V&T techniques and tools. This chapter presents information which assists in the selection of a complementary set of techniques and tools and their application throughout development. Specifically, it will discuss the following:

Types of analyses that are performed - description of static, dynamic, and formal analyses,

An integrated approach to performing V,V&T - combining the three types of analysis in a complementary fashion to achieve a V,V&T technology, and

Application of the V,V&T approach - the use of the three types of analysis in applying V,V&T to requirements, design, and code.

3.1 STATIC ANALYSIS

The static analysis detects errors through examination of the product, Figure 3.1-1. Some examples of the errors detected are: language syntax errors, misspellings, incorrect punctuation, improper sequencing of statements, and missing specification elements. Static analysis techniques may be manually or automatically applied, however, automated techniques require a machine readable product.

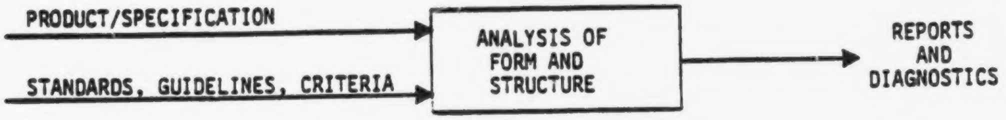

PRODUCT/SPECIFICATION
STANDARDS, GUIDELINES, CRITERIA
ANALYSIS OF FORM AND STRUCTURE
REPORTS AND DIAGNOSTICS

Figure 3.1-1 General Form of Static Analysis

Manual static analysis techniques may be applied to all development products, e.g., requirements statement, program code, or a users manual. In general, they are straightforward and, when applied with discipline, are effective in preventing and detecting errors.

The application of certain manual techniques, such as desk checking, inspection, and walkthroughs, provide certain advantages over the use of specialized automated techniques. One advantage is that different perspectives can be addressed simultaneously. A product may be examined for high level as well as detailed properties. Another advantage is that manual analysis provides an opportunity for the analyst to apply various heuristics, i.e. aids to discover errors, and subjective judgements. A general weakness of the manual techniques is that correct usage often involves tedious and repetitious activities. As the size of the application increases, the tendency is to compromise on the thorough application of the technique which results in an increasing chance of error.

Automated static analysis tools most often operate on program source code but can operate on requirements and design. Two kinds of static analyzers can be identified. The first gathers and reports information about a program. These kinds of analyzers generally do not search for any particular type of error in a program. A symbol cross reference generator is an example of this type.

The second kind of analyzer detects specific classes of errors or anomalies in a program. Examples of this type include: (1) parsers which determine the adherence of a program to the language syntax; they may include additional local programming conventions/standards such as program length; (2) techniques for analyzing the consistency of actual and formal parameter interfaces (see Figure 3.1-2); (3) techniques for comparing all variable references with their declarations to check for consistency; and (4) techniques for analyzing a program for erroneous sequences of events or operations, for example, attempting to read from a file before it is opened.

A much more detailed description of many specific techniques and tools can be found in the Technique and Tool Reference Guide[POWE].

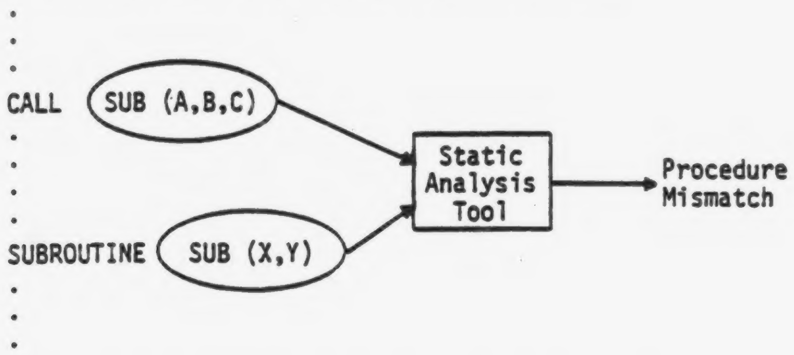


Figure 3.1-2 Module Interface Consistency Checks

3.2 DYNAMIC ANALYSIS

Dynamic analysis is the process of detecting errors through the study of the response of a program to a set of input data. It is usually accomplished through an automated simulation or the execution of a program, but may be manually performed, e.g., a walkthrough. Dynamic analysis (see Figure 3.2-1). is the processes of:

o preparing for test execution,
o test execution, and
o analysis of test results.

Preparing for test execution includes test data preparation and formulation of expected results.

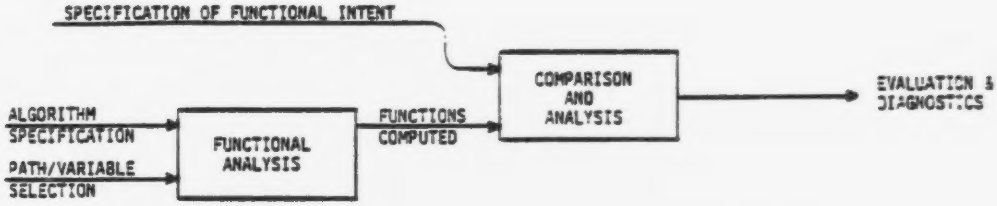


Figure 3.2-1 General Form of Dynamic Analysis

Test data preparation involves the formulation of test scenarios, test cases and the data which are to be entered into the program. Test scenarios and test cases are chosen as the result of analyzing the requirements and design specifications and the code itself. The test data should demonstrate and exercise externally visible functions, program structures, data structures, and internal functions including impossible and improbable test cases. Each test case includes a set of input data and the expected results. The expected results may be expressed in terms of final values and as statements (assertions), embedded in the software, about intermediate states of program execution.

The development of assertions takes place during the design and programming subphases of the lifecycle. In general, assertions are statements that specify the intent of a program's behavioral properties and constraints. Assertions about inputs, outputs, and intermediate steps of each function may be generated. A special notation (an assertion language) often is used to specify the assertions. Assertions are developed and inserted into the actual design specification and program code, usually as specially formatted comments.

Currently, test preparation is accomplished primarily through manual methods. These include definition of specification based functional tests and cause-effect graphing, a test case design methodology. Specification based functional testing is a method of developing test scenarios, test data and expected results through the examination of the program specifications (in particular, the requirements, design and program code). Test scenarios based upon the requirements have the objective of demonstrating that the functions, performance and interface requirements and solution constraints are satisfied. Test cases are determined from the design to test functions, structures, algorithms and other elements of the design. Test data are determined from the program to exercise computational structures implemented in the program code.

Cause-effect graphing [MYER] is a technique to develop test cases based upon inputs and input conditions. For each case, the expected outputs are identified. This technique operates on the requirements and design specifications.

Test execution involves running a program with the prepared test cases and then collecting the results. Testing may be planned and performed in a top-down or bottom-up fashion or a combination of the two. Top-down testing is performed in parallel with top-down construction in that a module is developed and tested while submodules are left incomplete as stubs or dummy routines. Bottom-up testing consists of testing pieces of code, individual modules, and small collections of modules in that order, before they are integrated into the total program. Bottom-up testing may require the development of test driver routines.

Test execution may be performed manually. Design and code walkthroughs (i.e., a manual simulation) provide a straightforward dynamic analysis method used prior to execution and during debugging.

Instrumentation, the insertion of code into a program to measure program characteristics, may be required to assist in testing. It may be done to capture intermediate values of a computation, measure execution frequencies during testing, or to detect assertion violations. Instrumentation may be done manually, or automatically as with test coverage analyzers and dynamic assertion processors.

Four types of tools are commonly used to assist in test execution. They are:

o Test coverage analyzers,
o Execution monitors,
o Dynamic assertion processors, and
o Performance monitors.

Test coverage analyzers capture and report execution details (e.g., statement execution counts). Some test tools (often built into a compiler or runtime system) monitor execution and check for the adherence to certain language semantics (e.g., array subscripts must remain within declared bounds, divisors must be non-zero). Dynamic assertion processors monitor program execution and report violations of the assertions supplied by the analyst. Performance monitoring tools report execution data describing execution frequency and timing information.

The process of analyzing the outputs of testing involves comparing the actual to expected results. This analysis requires a specification of the expected results for each case. Comparison of actual and expected results may be performed manually, or if the data are machine readable, an automated comparator may be used. The detection of assertion violations is normally accomplished through analyzing the assertion results generated by the instrumented program.

Dynamic analysis also includes determining the thoroughness of the testing. Commonly used metrics are the numbers (or percentages) of executable statements, branches, and paths actually exercised by the tests. Each succeeding metric subsumes the others. Since the number of paths grows exponentially with the number of decision points, even small programs can have many thousands of paths. For this reason, attempting to ensure that a high percentage of paths, exclusive of loop iterations, is exercised can be very costly. The statement and branch coverage metrics, although less conclusive, are most frequently used because they are relatively easy and inexpensive to implement.

3.3 FORMAL ANALYSIS

Formal analysis involves the use of rigorous mathematical techniques to analyze the algorithms of a solution. Algorithms are analyzed for numerical properties (e.g. accuracy, stability, convergence), efficiency, and correctness. At present, formal analysis is primarily a manual activity with limited automated assistance.

For more detailed information about techniques and tools for measurement of testing in the analysis of the numerical properties of algorithms, two types of analysis can be identified. The first type makes a conjecture about an algorithm's numerical properties and then proves a theorem to establish the conjecture. In the second type, an automated tool is used to analyze the numerical stability of a sequence of computations.

Complexity analysis first makes conjectures about the number of operations and/or the amount of space used on a problem of arbitrary but fixed size. The next step is to construct arguments which support the conjectures. There is considerable discussion as to how to measure or express software complexity. This technique has not reached maturity.

In another type of formal analysis, two basic approaches are used. One approach is to prove correctness of a whole program. The second approach is to prove correctness of particular program properties.

The basic strategy to both approaches is to construct a set of reasoning that shows that a solution specification satisfies its requirement(s). Typically, this is done by comparing the inferred transformations to the functional transformations dictated by the specifications of intent (see Figure 3.3-1).

To do the comparison, symbolic execution using selected control paths through the algorithmic specification is employed. For each path the values of each data object encountered are computed. Each value is not computed as a number but rather as a function or formula. Inputs are not assigned specific values but rather are treated as unquantified variables. As a result, at the end of tracing down the selected path, the functional relationships of outputs to inputs are determined. These functions or formulas are then compared to specifications of functional intent to see if the solution specification is correct as a value transformer.

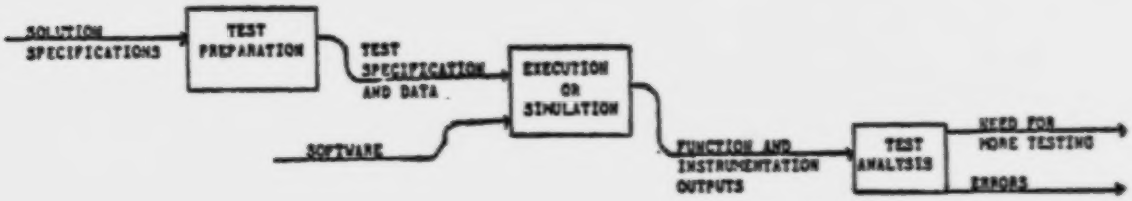


Figure 3.3-1 General Form of a Formal Functional Analysis

The formal analysis of a specific path can ensure that the functions computed in traversing the path are correct, and thus that any input processed by traversal of that path will necessarily be transformed into correct output values.

Formal analysis has two major limitations: strict dependence on the validity of the assumptions on which the analysis is based, and the complexity of the pathwise analysis of even small programs. Formal analysis has not yet reached its full potential as a V,V&T technique for several reasons. For one, program specifications are rarely written with sufficient precision to permit a rigorous comparison of intent with the implemented program. Also, accomplishing the analysis manually is extremely tedious and difficult (thus prone to error) and very few automated tools are currently available to facilitate effective application of formal techniques.

3.4 AN INTEGRATED APPROACH TO V,V&T

Each of the three types of analysis -- static, dynamic, and formal -- provides the V,V&T analyst with different types of specific information about the solution being examined. Static analysis focuses on the form and structure of the solution, but not the functional or computational aspects. Dynamic analysis addresses the functional, structural, and computational aspects. It is used to detect errors relating to these, but in practice is not used to demonstrate the absence of errors and is limited in demonstrating correctness. Formal analysis can provide a strong statement regarding certain properties of a solution including correctness, but is limited by the difficulty of application and lack of automated support.

These three types of analysis can be integrated to not only achieve the benefits of each, but to be complementary and provide a powerful composite V,V&T technology.

This section briefly describes a strategy to achieve this integration (see Figure 3.4-1). The following three sections of this chapter describe how this strategy can be applied to software requirements and design specifications, and to the code itself.

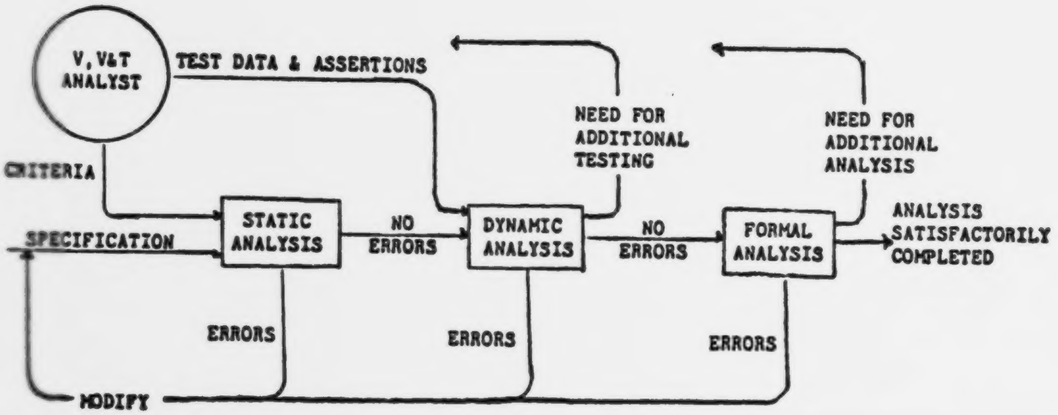

V,V&T ANALYST
TEST DATA & ASSERTIONS
CRITERIA
SPECIFICATION
STATIC ANALYSIS
NO ERRORS
DYNAMIC ANALYSIS
NEED FOR ADDITIONAL TESTING
NO ERRORS
FORMAL ANALYSIS
NEED FOR ADDITIONAL ANALYSIS
ANALYSIS SATISFACTORILY COMPLETED
ERRORS
ERRORS
ERRORS
MODIFY

Figure 3.4-1 General V,V&T Integration Strategy

The integration strategy is simple. First, static analysis techniques and tools are applied to analyze the form of the specification. These techniques and tools are straight forward to apply, applicable to all levels of specification, and identify flaws preventing the application of dynamic and formal techniques.

The second step is the application of dynamic analysis techniques and tools. These focus on the functional meaning of the solution and detect errors in their specification. These may be manually applied to the requirements and design specifications, with the code undergoing dynamic testing. This process, when applied with discipline, is effective, comprehensive and within the resource constraints of nearly all projects.

If additional assurances are required, the third step is the application of formal analysis techniques. These techniques can give strong assurances that the program design and code are fully traceable to the requirements and that they are a necessary and sufficient solution to the stated problem.

3.5 REQUIREMENTS V,V&T

Figure 3.5-1 illustrates the integrated V,V&T approach for requirements specification. It involves the application of static analysis techniques and tools to inspect the form and structure of the requirements and dynamic analysis techniques and tools to examine their functional aspects. For requirements specification, the definition of dynamic analysis is extended to include all actions which examine behavioral and performance aspects of requirements. Static and dynamic analysis techniques and tools, whether strictly manual, automated, or a combination of both are generally applicable to the four types of requirements:

o functional processing requirements,

- performance requirements,
- interface requirements, and
- solution (design) constraints.

Static techniques and tools are most useful for analysis of product form and interface requirements and solution constraints, while dynamic analysis (such as simulation) is most useful for performance requirements.

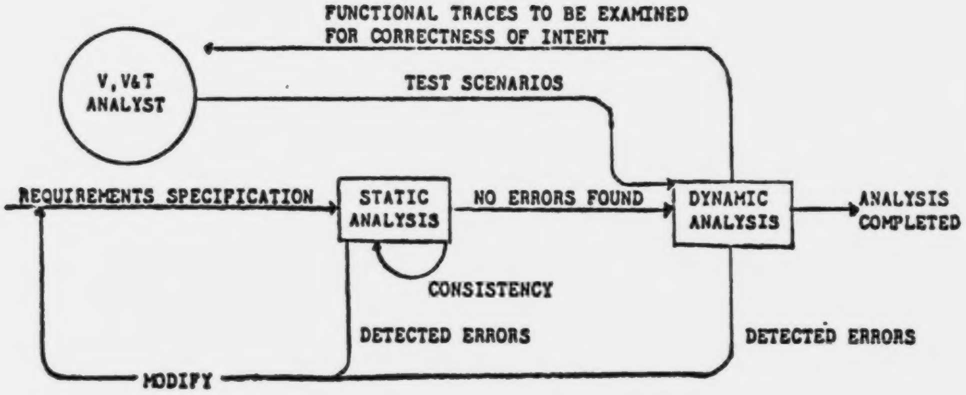

FUNCTIONAL TRACES TO BE EXAMINED FOR CORRECTNESS OF INTENT
V,V&T ANALYST
TEST SCENARIOS
REQUIREMENTS SPECIFICATION
STATIC ANALYSIS
NO ERRORS FOUND
DYNAMIC ANALYSIS
ANALYSIS COMPLETED
CONSISTENCY
DETECTED ERRORS
DETECTED ERRORS
MODIFY

Figure 3.5-1 Integrated Approach to Requirements V,V&T

Static analysis normally focuses on checking adherence to specification conventions, completeness and language syntax. Dynamic analysis at this level normally focuses upon information flows and functional interrelationships. Manual methods such as inspections, peer reviews, and walkthroughs are effective in accomplishing both types of analyses if rigorously performed. An example of using a manual method and dynamic analysis is to include input data as part of a walkthrough.

The application of both types of analysis depends upon the method used to specify the requirements. If the constructs of the specification scheme are clearly defined and capable of being represented in a computer processable form, then automated tools to aid in performing both the static and dynamic analysis may be used. Several such specification methods with supporting tools are in existence, but not widely used or available.

The requirements specification methods commonly in use today support the representation of information aggregates and functional capabilities in a hierarchical form. This allows for two types of checks of the specification for internal consistency. The first examines the decompositions performed in creating the functional and information hierarchies. Each level of decomposition is checked to ensure that it is consistent with the previous. The second type of analysis examines the consistency between the functional and information views. Information aggregate specifications usually include indications of the functions which generate and utilize them. Conversely, function specifications usually include indications of the information aggregates which they generate and utilize. These two types of specifications are readily checked for consistency.

The dynamic analysis of a specification often involves the evaluation of the functional aspects of the solution with respect to the intent as documented in previous specifications. This is not possible because the requirements statement is the first formal specification. The only previous document to which it might be compared is the project request from the initiation phase. Normally, this is an insufficient basis for requirements analysis.

Consequently, the dynamic analysis of requirements is accomplished through the examination of expected functional behavior to determine if it will solve the problem. Exercising information transformations through plausible sequences of the required functions will provide insight to the expected behavior. This is usually guided by scenarios which describe expected use, including input values and expected results. (These scenarios form the basis for the software test data and are refined and complemented in the design and coding subphases.) This type of analysis aids in the recognition of errors, oversights, and contradictions in the requirements.

3.6 DESIGN V,V&T

Figure 3.6-1 illustrates the integration strategy as applied to a design specification. Similar to the requirements analysis, static analysis techniques and tools are applied to analyze the form and structure of the design specification(s). Dynamic analysis techniques and tools are used to examine the functional aspects of the design. In addition, formal analysis may be applied to rigorous design specifications to gain additional assurances of correctness relating to certain functional properties.

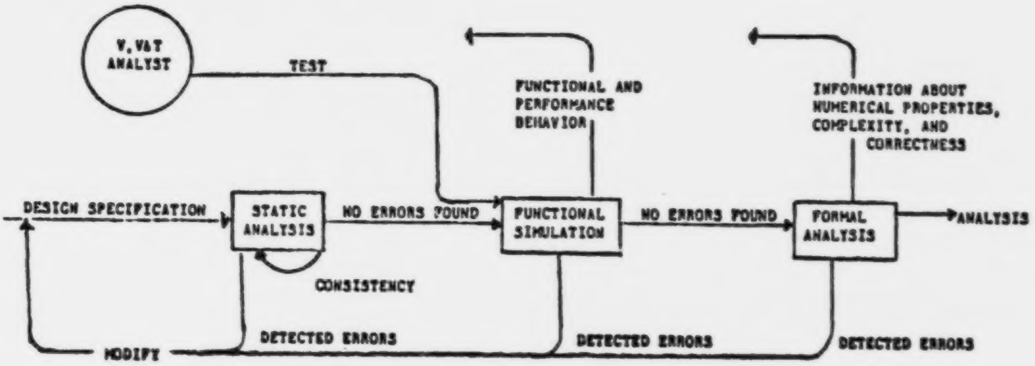

V,V&T ANALYST
TEST
FUNCTIONAL AND PERFORMANCE BEHAVIOR
INFORMATION ABOUT NUMERICAL PROPERTIES, COMPLEXITY, AND CORRECTNESS
DESIGN SPECIFICATION
STATIC ANALYSIS
NO ERRORS FOUND
FUNCTIONAL SIMULATION
NO ERRORS FOUND
FORMAL ANALYSIS
ANALYSIS
CONSISTENCY
MODIFY
DETECTED ERRORS
DETECTED ERRORS
DETECTED ERRORS

Figure 3.6-1 Integrated Approach to Design V,V&T

The basis for design V,V&T is the design documentation. The mechanisms or design representation schemes used to specify the design determine the specific analysis techniques which can be employed. The degree of formality of the schemes used determines the need for and ability to perform static analysis. The content of the information captured by the scheme determines the dynamic and formal analysis techniques which may be applied. If the schemes can be translated into a computer processable form, then automated techniques may be used.

Syntactic and semantic errors can be detected by the static analysis of a design specification. The syntactic errors are errors in form rather than content. These are not the primary emphasis of static analysis. By assuring the correct form and structure of the design specification, the way is cleared for more in depth analysis of the semantic content of a specification.

The semantic errors which can be detected in a design involve information or data decomposition, functional decomposition, control flow, and data flow. Selected examples of these are discussed below.

Design specification schemes generally provide mechanisms for specifying algorithms and their inputs and outputs in terms of modules. Various inconsistencies in specifying the flow of data objects through the modules are possible. For example, a module may need a particular data object which no other module creates. Conversely, a module may create a data object which is not input to any module. Static analysis can be applied to detect these types of data flow errors.

Certain errors made during the composition of a design can also be detected. Design specifications are usually created by iteratively supplying detail. Thus, most schemes facilitate the hierarchical expression of a design. Data aggregates and functional modules may be specified in terms of their gross overall characteristics and then specified in more detail. A hierarchical specification structure is regarded as an excellent vehicle for expressing and understanding a design. It does, however, leave open the possibility of inconsistencies between levels of detail. For example, the inputs and outputs specified for a high level module must be equivalent to the cumulative inputs and outputs of the submodules. Any inconsistencies indicate an error in the evolving solution. Static analysis can determine the presence or absence of such errors.

Dynamic analysis of a design is generally accomplished by some form of design simulation. This may be a manual walkthrough or simulation using a model of the design. A design walkthrough is similar to the analysis performed on the requirements except at a greater level of detail. It is guided by usage scenarios which are refined and expanded from those used in the requirements analysis. At the design stage, test cases are developed and used during the walkthrough to exercise all functions. During detailed design, test cases are developed and used to examine the software structure as well as its functions. (These test scenarios and test cases are used during the programming subphase for testing the code.) Manual walkthroughs, when rigorously performed and guided by documented test scenarios, are a cost effective technique for analyzing a software design.

For larger software designs and highly critical systems or components, an automated simulation may be appropriate. This requires the construction and execution of a solution model with the test scenarios. The model is validated as a faithful representation of the solution. The cost of simulation generally increases with the complexity of the model and the degree of model fidelity. Thus, model simulation is only used when it can be cost justified.

Formal analysis techniques may be manually applied to a design specification This involves tracing paths through the design specification and formulating a composite function for each. This procedure is more feasible at higher levels of a hierarchical design specification because less detail is present, resulting in algorithm paths being relatively short and few in number. Thus, the evolved functions remain concise and manageable.

The purpose for deriving these composite functions for a given level of design is computed and compared to the functions of the previous level. This process assures that the design is continuing to specify the same functional solution as it is hierarchically elaborated.

The formal analysis of a design specification can be improved by using automated symbolic execution tools. Such tools can be expensive to create and operate. In return, however, they offer greater speed and capacity for manipulating detailed specifications. Thus, the functional effects of all levels of a design specification can be determined.

Unfortunately, such sophisticated tools are rarely applied in contemporary practice principally due to the novelty of such tools, the infrequent use of rigorous design formalisms, and expense. It is likely, as experience with these tools is gained and as better design representation schemes emerge, their use will increase.

3.7 CODE V,V&T

The third and last application of the general V,V&T integration strategy is to program code. This is portrayed in Figure 3.7-1. The V,V&T procedure for code includes both static and dynamic analysis. It may also encompass formal analysis.

Because code is written to be compiled and executed, it is necessarily written in a computer language having defined syntax and semantics. Syntax rules are generally enforced through the use of a compiler. Unfortunately, most compilers do not carry out checking for many important semantic errors, e.g. array range subscript checking, type analysis or routine interface analysis.

Static analysis techniques and tools are used to ensure the proper form of programming products such as code and documentation. This can be accomplished by checking adherence to coding and documentation conventions, interface and type checking, etc. The checking can be done by manual techniques such as inspections and automated tools such as a code auditor.

Next, dynamic analysis techniques are employed to study the functional and computational correctness of the code. Initially, manual techniques such as walkthroughs can be used as an effective forerunner to testing. These techniques can focus on modules to complement the role of testing.

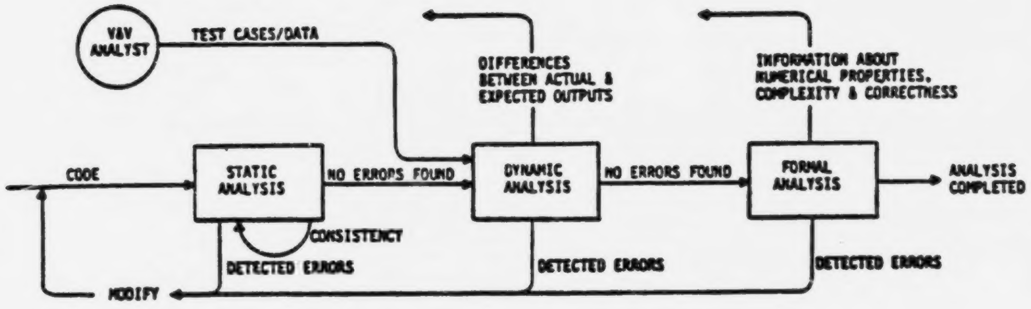

V&V ANALYST
TEST CASES/DATA
DIFFERENCES BETWEEN ACTUAL & EXPECTED OUTPUTS
INFORMATION ABOUT NUMERICAL PROPERTIES, COMPLEXITY & CORRECTNESS
CODE
STATIC ANALYSIS
NO ERRORS FOUND
DYNAMIC ANALYSIS
NO ERRORS FOUND
FORMAL ANALYSIS
ANALYSIS COMPLETED
CONSISTENCY
DETECTED ERRORS
DETECTED ERRORS
DETECTED ERRORS
MODIFY

Figure 3.7-1 Integrated Approach to Code V,V&T

Testing is accomplished by running the code on the test data sets which were developed during the requirements and design subphases. As emphasized earlier, the correctness of the test executions is determined more definitively when the expected results are specified. Testing for adherence to assertions is also highly advisable. The assertions, which are products of the design activity, provide additional information regarding expected behavior of the software.

In cases where software is being developed in an environment other than the production environment, testing is more problematical. Here the production environment can be simulated or taken into account informally. In any case, the validity of the test results depends upon the fidelity of the simulation or informal judgements. If there is a significant difference in the two environments, there will be an eventual need for some additional testing in the actual production environment. The balance between simulation testing and actual production environment testing must be determined for each individual project, based partially upon the availability and expensiveness of the production environment.

If assurances of correctness over and above those provided by dynamic analysis are required, then formal analysis should follow testing. For most projects, this simply takes the form of inspections to see that the various algorithms dictated by the design have been correctly implemented. Coming after a battery of successful tests, this activity needs to focus upon algorithms which are deemed crucial and yet inadequately tested (perhaps due to high cost of exercising them). Of course, some applications may be particularly critical in nature and demanding of very great assurance of correctness. Symbolic evaluation and other formal analysis methods can be effective in achieving such levels of confidence, but the cost is high, generally entailing development of special purpose evaluation tools.

3.8 SUMMARY

It is reasonable to conclude that for any software project, the integration strategy for V,V&T activities will result in a better return on investment of costly resources and increased confidence in the desired qualities of the final product. Although the strategy is sound and is strongly endorsed, it does not address the problem of how to select and configure specific techniques and tools for a specific project. The problems in doing this appear to be of two major types:

- o Projects vary in many respects and this variation strongly affects the proper selection of techniques and tools, and
- o There are a multiplicity of techniques and tools but there is little guidance in their selection to meet specific project needs.

The first problem is addressed in Chapter 4 of this document. The second problem is addressed in part by the report "Features of Software Development Tools" [HOUGa] and the "NBS Software Tools Database" [HOUGb].

In time, a reasonable complement of proven V,V&T support techniques and tools will evolve. Then, with a judicious selection and systematic, integrated application during all lifecycle phases, it will be possible to effect significant reductions in the cost of software production as well as improvements in the quality of developed systems.

CHAPTER FOUR

V,V&T PLANNING

An important part of problem solving is planning. This is certainly true in software development. Within this context, V,V&T planning is an integral part of the overall project development planning. It is started during the initiation of a project along with all other planning.

The objective is to establish a V,V&T plan to suit the needs of the project. Software development and maintenance projects vary with respect to many factors such as size, criticality, complexity, etc. These factors must be taken into account so that the plan is both feasible and effective.

This chapter describes the contents of a V,V&T plan and presents a process for developing one. Section 4.1 explains how V,V&T planning is part of the overall planning process. It briefly describes the outcome of a V,V&T plan and introduces a four step process to assist in developing part of the plan. The remaining sections present each of the four steps.

Step I - Identify V,V&T Goals - The result is a set of specific measurable goals of the validation, verification, and testing activities of the project.

Step II - Determine Influences on V,V&T Activities - The result is a set of factors to be taken into account in the planning of the V,V&T activity.

Step III - Select V,V&T Techniques and Tools - The outcome is a list of V,V&T practices, techniques and tools to satisfy the identified goals within the constraints of the environment.

Step IV - Develop a Detailed V,V&T Plan - The result is a phase-by-phase V,V&T plan specifying V,V&T practices, as well as specific tasks, schedules, and budgets.

4.1 V,V&T PLANNING IN THE TOTAL PROJECT CONTEXT

Project planning is one of the major functions of project management. V,V&T planning is one part of project planning. Figure 4.1-1 illustrates this relationship and examples of planning products.

The objective of planning is to document a plan of action. It may include: objectives, approach, schedule, and allocated resources. A plan is used to initiate and control a project.

Figure 4.1-2 shows the evolution of a project plan. A project is formed to solve a problem. This first step in project formation is to define its mission or its objectives. Next, an approach is formulated. This is refined with additional details about the approach and documented in the project plan.

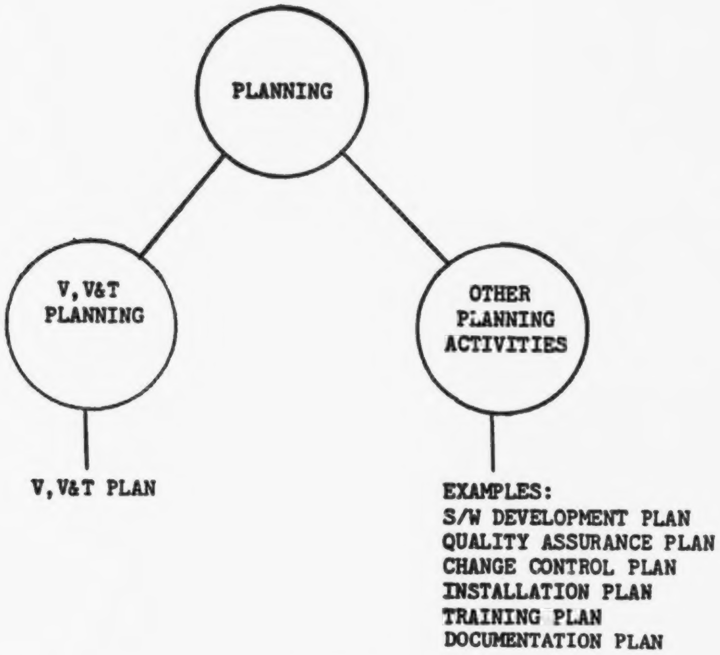

PLANNING
V,V&T PLANNING
OTHER PLANNING ACTIVITIES
V,V&T PLAN
EXAMPLES:
S/W DEVELOPMENT PLAN
QUALITY ASSURANCE PLAN
CHANGE CONTROL PLAN
INSTALLATION PLAN
TRAINING PLAN
DOCUMENTATION PLAN

Figure 4.1-1 Software Project Planning

Problem → Mission → Approach → Plan

Figure 4.1-2 Evolution of Project Plan

This chapter presents a V,V&T planning process which follows a similar set of steps. The interactions between the general and the V,V&T planning activities are important. The general planning activity will drive the V,V&T planning activity, which in turn will provide input and feedback to the first. Some of the important interactions are:

- V,V&T goals are established which complement the general project approach.
- V,V&T techniques and tools will assist in achieving the V,V&T goals only if they are integrated with the project approach.
- The details of the V,V&T plan, e.g., time and resource requirements, must be factored into the overall schedule and budgets.

Planning activities, including the preparation of the V,V&T plan, may begin in the initiation phase but are completed early in the development phase (Figure 4.1-3). Plans specifying tasks, budgets, schedules, etc. for the requirements and preliminary design subphases are completed early in the subphase. Plans specifying information for the subsequent subphases are also prepared to the level of detail possible. These, though, are likely to be revised as the technical scope, size and complexity of the solution are defined in the preliminary design activity. Special attention must be paid to activities that require long lead time, or need to begin early in the project, e.g., tool acquisition or development, personnel hiring or training.

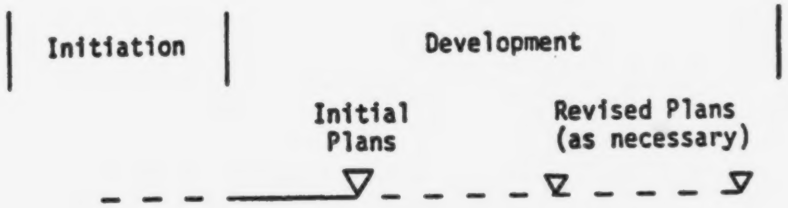


Figure 4.1-3 Preparation of Plans

The result of the V,V&T planning activity is a documented V,V&T plan. The complexity of the plan and the effort required to prepare it will depend upon the size and nature of the project. More effort and a greater level of detail may be required as the size, complexity, and critical nature of the project increases.

A V,V&T plan specifies goals of the project's V,V&T activities relating to the products of each phase. For these products, the activities and supporting techniques and tools are then described which make up the approach to achieve the goals. An abbreviated outline of a V,V&T plan is presented in figure 4.1-4. A detailed outline is presented in Section 4.5.

The remainder of this chapter describes the four step approach to preparing a V,V&T plan. Each step and substep are presented in a standard format, including the following items as appropriate:

- General Description
- Outputs
- Inputs
- Interrelationships with Other Steps
- Roles and Responsibilities
- Method and Supporting Techniques
- Worksheet
- Example
- Comments

The worksheets are suggested as documentation aids. Examples are presented using these worksheets.

For small projects it may be sufficient to abbreviate the process, following it as necessary to achieve the ultimate objective, a V,V&T plan. For larger projects, the process provides a systematic method for developing a comprehensive V,V&T plan. In either case, the outcome should suit the goals of the project.

I. BACKGROUND AND INTRODUCTION

II. V,V&T GOALS

1. Summary of V,V&T Goals and Measurement Criteria
2. References and Related Documents

III. PHASE BY PHASE PLANS

For each phase, information including:

1. V,V&T Goals for Each Product
2. V,V&T Activities
3. Techniques and Tools
4. Assumptions and Other Information
5. Roles and Responsibilities
6. Schedules
7. Budgets
8. Personnel

Figure 4.1-4
ABBREVIATED OUTLINE OF A PROJECT V,V&T PLAN

4.2 STEP I: IDENTIFYING V,V&T GOALS

General Description: V,V&T goals are formulated from software requirements, and, secondarily, software product standards. V,V&T goals address the functional, structural, and computational correctness of the software, the correctness of the form, its performance, and other attributes such as reliability and portability. The achievement of these goals demonstrates that the software has the desired attributes. This is straightforward in some instances, e.g., demonstrating the correct operation of a tape sort/merge capability, while in others it may be a matter of degree, e.g., clarity and understandability of documentation. The term goal describes a specific, measurable outcome. A complete definition of a goal includes the statement of the measurement criteria to determine that the goal has been reached.

Goals are established to define tangible results of the activity (e.g., a report) or a culminating event (e.g., a review). Measurement criteria specify how to determine that the result is satisfactory.

Goals relating to products (intermediate and final) are established which pertain to either their form or the content. The goal will precisely describe the desired product attributes. This distinction will be useful in selecting techniques and tools.

Goals relating to an activity or product should be examined as a set, and categorized as high, medium, or low in importance. This information will be used in Step III if trade-offs (because of time, budget, or resource constraints) are necessary. This examination also acts as a re-evaluation point, and may result in the revision, elimination, or combination of goals. The worksheet examples for steps I and III are separate. To eliminate duplication, they can be combined into one worksheet.

Outputs of Step I:

- A set (one or more) of goals, measurement criteria, and importance of each V,V&T goal

Inputs to Step I:

- Project Proposal
- Statement of Problem/Needs
- Statement of Software Requirements
- Applicable Product Standards

Interrelationships with Other Steps: The list of goals from this step will be used in Step III. This step can be done parallel with Step II.

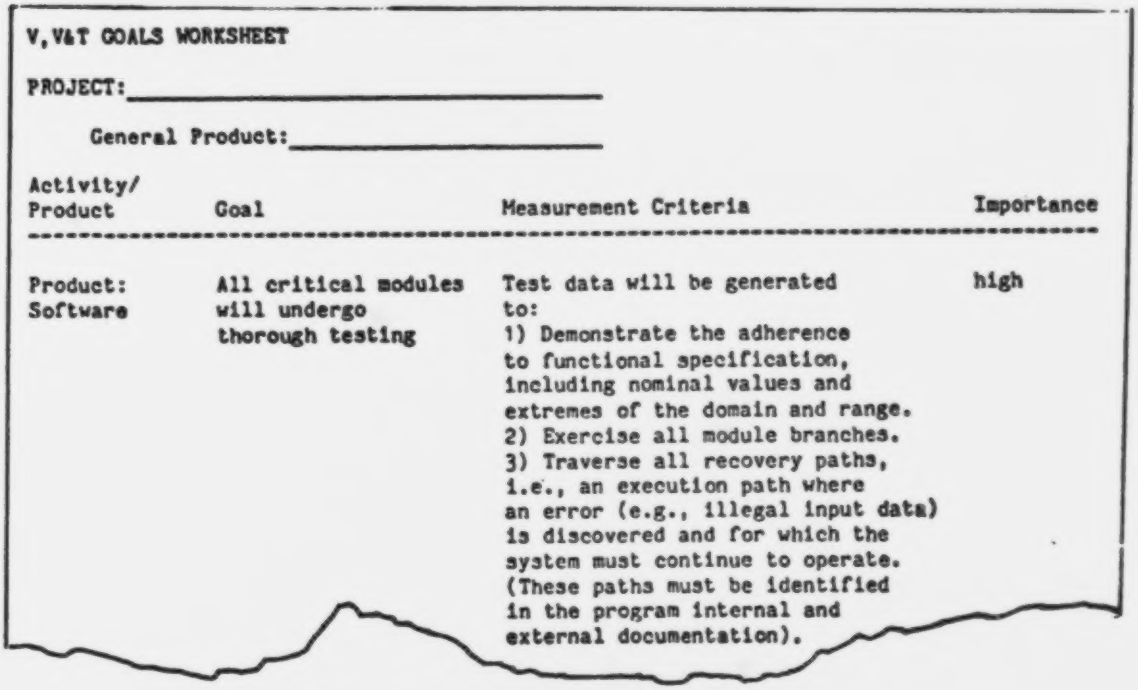
Roles and Responsibilities:

- User/Customer - to identify the goals and measurement criteria to ensure that the product is acceptable

- Development Staff and Management - to ensure that V,V&T goals and measurement criteria are both measurable and feasible and reflect existing standards and conventions concerning product quality

Methods and Supporting Techniques:

- Standards, guidelines and conventions that contribute to product quality
- Requirement analysis and specification aids that support the identification and documentation of the software requirements
- Checklists for identifying V,V&T goals based upon the software requirements
- Inspection, walkthrough, review procedures that support the analysis and evaluation of the V,V&T goals



V,V&T GOALS WORKSHEET

PROJECT:____________________________

General Product:____________________

Activity/ Product	Goal	Measurement Criteria	Importance
Product: Software	All critical modules will undergo thorough testing	Test data will be generated to: 1) Demonstrate the adherence to functional specification, including nominal values and extremes of the domain and range. 2) Exercise all module branches. 3) Traverse all recovery paths, i.e., an execution path where an error (e.g., illegal input data) is discovered and for which the system must continue to operate. (These paths must be identified in the program internal and external documentation).	high

Figure 4.2-1 Goals and Measurement Criteria

Example: Figure 4.2-1 identifies the goal, measurement criteria, and importance of testing critical modules.

Comments: A worksheet which assists in specifying goals and selecting the associated techniques and tools are described in the following sections. It can be used as an aid in documenting the results of this step.

- V,V&T Goals Worksheet - used in Step I to define a specific set of goals and the measurement criteria

- V,V&T Technique and Tool Selection Worksheet - used in Step 3 to select the techniques and tools to attain each goal

4.3 STEP II: DETERMINE FACTORS INFLUENCING V,V&T ACTIVITIES

General Description: This step involves collecting a mixture of both objective and subjective information (e.g., the attitudes of the technical staff concerning V,V&T). This information will, in some cases, lead to clear cut decisions regarding the implementation of a given technique or tool. In other instances, it may only indicate potential areas of difficulty or supply information to support a decision intuitively.

Four general areas in a suggested order of investigation are presented below.

Software V,V&T and development technology
Project groups and roles
Project constraints
Available computing resources

These areas of investigation are further described below.

Outputs of Step II:

- o An assessment of available software engineering and V,V&T technology
- o The capabilities and expected involvement of various groups
- o The budget and schedule constraints upon the V,V&T activities
- o An inventory of computing resources.

Inputs to Step II:

- o Software engineering standards, guidelines, procedures, development and maintenance methodologies, etc.
- o Information regarding technical personnel
- o Information regarding project budget and schedule
- o Information regarding computing resources

Interrelationships with Other Steps: This step can be performed in parallel with Step I - The Identification of V,V&T Goals. It provides inputs into Step III and Step IV.

Roles and Responsibilities: This activity is primarily the responsibility of the development organization.

Methods and Supporting Techniques: The following is a checklist of the four areas of investigation mentioned above.

Software V,V&T and Development Technology

- o Software V,V&T (see techniques and tools descriptions in [POWE])
  - techniques
  - tools
  - standards and guidelines
  - conventions and procedures

- o Software development

  - methodologies
  - specification techniques and design representation schemes
  - techniques and tools
  - documentation standards and guidelines
  - technical assistance
  - training

Project Groups and Roles (The following groups and their expected roles should be balanced against their knowledge, experience, attitudes, and expectations.)

- Customer
- User
- Technical staff
- Management (e.g., project, or organizational, and support, e.g., legal)
- Independent groups (e.g., quality assurance, independent V,V&T)

Project Constraints

- Schedule of major project results
- Budget information for:
  - analysis and review of each product
  - acquisition and/or development of support capabilities and tools
  - technical assistance and/or training in the use of selected techniques and tools
  - use of independent testing, V,V&T and quality assurance teams

Available Computing Resources

- Available machines
- Access methods and devices
- Support utilities
- Technical support

4.4 STEP III: SELECT V,V&T TECHNIQUES AND TOOLS

General Description: This step can be divided into two activities:

- Establish a candidate list of techniques and tools to be used to reach each goal, and
- Select and evaluate the techniques and tools from the candidate list to be used.

One input to the first activity is the list of V,V&T goals. Goals should have been established for: a) activities by phase/subphase, and b) the form and content of each product. The other input to this activity is information about each candidate technique or tool.

Candidate techniques and tools may be chosen from three sets: a) those commonly used on software development projects (information on these was collected during Step II); b) those used elsewhere which may be acquired; and c) those which can be developed by the project staff.

For each candidate technique or tool, some basic information is needed. This includes the availability of documentation, training, consulting support, and tool effectiveness. Also required are cost, time required for acquisition, and need for training project personnel.

Once the list of candidate techniques and tools has been created for a given goal, the selection activity begins. The feasibility of implementing each technique or tool is studied. The implementation requirements (e.g., personnel resources and level of expertise) are compared against the constraints identified in Step II (e.g., personnel experience). This identifies inappropriate candidates. The techniques and tools to be used are chosen from the remaining.

The assumptions made in the selection process need to be documented for use in Step IV. For example, assumptions regarding the acquisition, implementation, application, and required training should be recorded.

Outputs of Step III:

- A list of V,V&T techniques and tools
- Assumptions made in selection process

Inputs to Step III:

- V,V&T goals
- Information on candidate tools and techniques
- Influences to be considered during selection (from Step II)

Interrelationships with Other Steps: The list of V,V&T goals may be revised because one or more goals are determined to be infeasible.

Roles and Responsibilities: The development staff have primary responsibility for this step. It should be performed by senior staff familiar with both the management and the technical aspects of the project. Generally, a representative of the groups who will use the techniques and tools should be involved in the selection process.

Methods and Supporting Techniques: The "Validation, Verification, and Testing Technique and Tool Reference Guide" [POWE] describes individual techniques and tools.

Worksheet and Example: Figure 4.4-1 shows the V,V&T Technique and Tool selection worksheet. Figure 4.4-2 shows a completed worksheet for a goal from the previous example.

V,V&T TECHNIQUE AND TOOL SELECTION WORKSHEET

PROJECT: ______________________

Goals	Measurement Criteria	Candidate Techniques & Tools	Rationale for Choice/Elimination	Final Choice & Comments
The goals and criteria established during Step I.		Techniques or tools which may potentially be used in reaching each goal are listed in this column	Each technique or tool will be analyzed and either chosen or rejected. The decision is documented here.	The final choice(s) will be listed here. Also any information pertaining to the use or application should be stated here or reference to supplementary material given. It is important to note here any special information that should be taken into account in the detailed planning (Step IV) and/or implementation of the technique or tool.

Figure 4.4-1 V,V&T Technique and Tool Selection Worksheet

V,V&T TECHNIQUE & TOOL SELECTION WORKSHEET

PROJECT:____

Goals	Measurement Criteria	Candidate Techniques & Tools	Rationale for Choice/Elimination	Final Choice & Comments
For critical functions: All critical modules will undergo thorough testing	The functions of these modules must be tested for extremal values of the domain and range, as well as on nominal test data: 1) All branches will be tested, including all 'recovery' paths, i.e. an execution path where an error (e.g., illegal input data) is discovered and for which the system must continue to operate (these paths must be identified in the program internal and external documentation).	-Data Flow Analyzer	Can be used to check out programs from the data flow perspective. Availability is uncertain.	Investigate availability
		-Assertion Generation and Assertion Processing	Assertion generation will aid in specification documentation and test generation. Assertion checking via an autimatic processor will aid in testing and test coverage measurement. Availability is uncertain.	Investigate availability
		-Specification Based Functional Testing	This technique can be used in the requirements and design phases to incrementally build test sets.	Used in conjunction with test set documentation standards and test libraries to store data and results.
		-Test Coverage Analyzer	This tool can summarize test coverage to enable an accurate assessment of the thoroughness of the testing.	Will be used. Modifications to existing tool will will be necessary.

Figure 4.4-2 Completed V,V&T Technique and Tool Selection Worksheet

4.5 STEP IV: DEVELOP A DETAILED V,V&T PLAN

General Description: The preceding three steps provide a definition and means of attaining V,V&T goals. The detailed V,V&T plan can now be formulated by defining each activity in terms of specific tasks and outcomes. The roles and responsibilities of the groups involved, (e.g., customer, user, technical, management) are defined; sign-off procedures at milestones are established. Underlying assumptions are stated, and interrelationships with other activities (preceding, parallel with, and succeeding) are defined. Budget allocations and schedules are established. Plans also include training and tool acquisition or development activities.

Outputs of Step IV:

- o A Detailed V,V&T Plan

Inputs to Step IV:

- o List of V,V&T techniques and tools
- o Project Schedule, budget, and personnel information
- o V,V&T goals
- o Technique and tool selection assumptions

Interrelationships with Other Steps: The plan is built on the outputs of prior steps.

Roles and Responsibilities: This involves both management and technical staff. Management approval is required for schedules and budgets. Customers/users approve their role.

Methods and Supporting Techniques: Presented below is an annotated outline of a general V,V&T plan. This can be used as a checklist.

An Outline of a Project V,V&T Plan

I. Background and Introduction

The purpose of this section is to establish the context for the document. It should be brief and introductory in nature. It should focus on those aspects of the problem and/or solution which are the sources of the goals for V,V&T. Where additional information is necessary, it can be included or referenced.

A. Statement of Problem
B. Proposed Solution
C. Project Summary
D. References/Related Documents

II. V,V&T Goals and Measurement Criteria

This section presents the results of Step I. It presents the project V,V&T goals, measurement criteria, and importance. The exact format and content of this section may vary. If the worksheets are used, then they may be included. Alternatively goals could be summarized as presented below. A third alternative would be to state the project level information in this section and present all phase (and product) specific information in the next section.

A. V,V&T Requirements
  A.1. Functional
  A.2. Performance
  A.3. Reliability
  A.4. Other

B. V,V&T Goals and Measurement Criteria for each goal
  B.1. General
  B.2. Product specific
  B.3. Phase specific

C. References/Related Documents

III. Phase by Phase V,V&T Plans

This section contains the description of project V,V&T practices. Section A defines the project approach: phases, their products, and the major reviews and check points. Where a practice is common to all phases it may be described here. Section B and later sections describe phase specific activities.

A. Project Background and Summary Information
  A.1. Project Phases and Products
  A.2. Major Reviews (both management and technical)

B. Requirements Subphase V,V&T Activities
  B.1. Summary of V,V&T Goals
  B.2. V,V&T Activities
  B.3. V,V&T Techniques and Tools Selected
    a. Reviews
    b. Methods of Analysis
  B.4. Support Requirements and Assumptions
  B.5. Roles and Responsibilities
  B.6. Schedules
  B.7. Budgets
  B.8. Personnel

C-F. Preliminary Design through Installation respectively (same format as B).

Appendix A Project and Environmental Considerations

This section contains the information which was compiled during

Step II.

A. Technical Issues
B. Organizational/Personnel Issues
C. Project Budget and Schedule
D. Computing Resources

Appendix B Technique and Tool Selection Information

This section contains the work sheets of Step III.

A. Goals, and Related Techniques and Tools
B. Worksheets

CHAPTER FIVE

EXAMPLE APPLICATIONS OF VALIDATION AND VERIFICATION TECHNOLOGY

This chapter presents a series of examples in which the concepts of software development, software V,V&T, and V,V&T planning are illustrated. The purpose is to show how these concepts may be applied in a variety of situations.

Four examples are presented, using an automobile insurance transaction processing system as the system being developed and maintained. The first three examples describe development activities in differing environments and the fourth describes selected maintenance activities.

The four examples are structured recognizing that programming environments present in government and industry settings vary significantly in resources available to support V,V&T. The transaction processing system is considered first in an environment where only manually applied techniques are employed. The progressive benefits of utilizing automated and more advanced techniques can then be seen by studying how V,V&T is done on the same application, first through adding a small number of automated tools to the set of manual techniques, and lastly through use of a full complement of modern automated tools.

5.1 OVERVIEW OF EXAMPLES

Examples 2, 3, and 4 build upon Example 1. The tools introduced in Example 2 are to be used in addition to the manual techniques described in the first example. The discussion in each example centers on the additional capabilities introduced. Figure 5.1-1 presents an overview of the different V,V&T tools and techniques which are used in the four examples.

The software development subphases for each example are:

- o Requirements,
- o Preliminary design,
- o Detailed design, and
- o Programming (includes testing).

Figure 5.1-2 shows the information flow and relationships among the four subphases of the software development lifecycle depicted in the examples.

Each of the examples will be presented showing for each phase:

- o Inputs to the phase,
- o Outputs from the phase,
- o Supporting technology used in the phase, and
- o Activities which comprise the phase.

	Software Development			Maintenance
	Example (#1)	(#2)	(#3)	(#4)
		Tool Complements		
	Manual Techniques	Minimal	Extensive	
Supporting Technology	Graphical Requirements Representation	Requirements & Design Languages		Problem Reporting and Change Request Mechanisms
Static Analysis	Walkthroughs Formal Reviews	Cross-Referencer Requirements & Design Language Analyzers	Interface Checker Dataflow Analyzer Standards Checker	Ad Hoc
Dynamic Analysis	Functional Testing	Test Coverage Analyzers File Comparator	Assertion Generation Assertion Checking	
Formal Analysis				

Figure 5.1-1 Overview of Examples

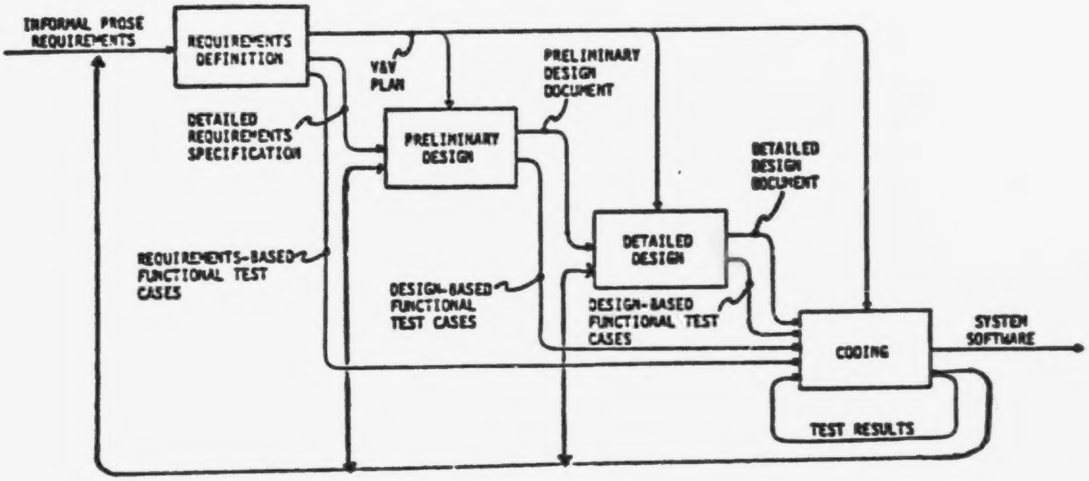

INFORMAL PROSE REQUIREMENTS
REQUIREMENTS DEFINITION
V&V PLAN
PRELIMINARY DESIGN DOCUMENT
DETAILED REQUIREMENTS SPECIFICATION
PRELIMINARY DESIGN
DETAILED DESIGN DOCUMENT
DETAILED DESIGN
REQUIREMENTS-BASED FUNCTIONAL TEST CASES
DESIGN-BASED FUNCTIONAL TEST CASES
DESIGN-BASED FUNCTIONAL TEST CASES
CODING
SYSTEM SOFTWARE
TEST RESULTS

Figure 5.1-2 Example Software Development Lifecycle

Most activities will contain:

- o V,V&T purpose for the activity,
- o V,V&T technique(s) used by the activity, and
- o Example(s).

Tables 5.2-1, 5.3-1, 5.4-1, and 5.5-1 provide a summary of the development and V,V&T techniques and activities for the manual, minimal automation, full automation, and maintenance activities. These tables present a synopsis of the examples presented in this chapter.

The application area used in the examples is representative of a large number of government and commercial systems. Transaction processing systems are perhaps the most common of all commercial systems. Many banking, billing, payroll, inventory, and insurance applications are in this category. Thus, the four examples focus on this area.

The transaction processing system is set in the context of an auto insurance application. In order to limit the size of the presentations some simplifications have been made in the application area. An expert in the auto insurance field will surely detect omissions and simplifications in details of the system as described. The reader is encouraged, however, to not focus on the application area, but rather on the V,V&T principles applied. The details provided enable presentation of specific instances of the application of V,V&T techniques.

The Auto Insurance Management System (AIMS) described in the examples supports all the major activities of such a company: accounts payable (claims processing), accounts receivable (premium processing), management reports, and database management. AIMS must issue client premium due notices, checks to repair shops (or clients), recommend policies that should be cancelled, monitor the company's day-to-day financial health, and so forth. Further details of the system's requirements are included in the first example.

5.2 EXAMPLE 1: SOFTWARE DEVELOPMENT USING MANUAL V,V&T TECHNIQUES

In this example the details of the AIMS are presented in addition to the actual manual V,V&T practices which are applied within each of the four phases of the software development lifecycle.

TABLE 5.2-1
EXAMPLE 1 SUMMARY
SOFTWARE DEVELOPMENT USING MANUAL V,V&T TECHNIQUES

SURPHASES	REQUIREMENTS	PRELIMINARY DESIGN	DETAILED DESIGN	PROGRAMMING
INPUT	o Informal prose Requirements	o Detailed Requirements Specification - Revised Prose Description - Revised Graphical GR Representation o V,V&T Plan	o Preliminary Design Document o V,V&T Plan o Test Cases	o Detailed Design Document V,V&T Plan o Test Cases
OUTPUT	o Detailed Requirements Specification o V,V&T Plan o Initial Test Cases	o Preliminary Design Document - Further Refined GR System Representation - Detailed User Input/Output Specification - Basic Control Flow Design - Basic System Information Specification o Additional Test Cases	o Detailed Design Document o Additional Test cases	o System Software o Test Results
SUPPORTING TECHNOLOGY	o Formal Requirements Reviews o A Graphical Requirements Representation Method o Requirements-based Functional Testing	o Reviews o A Graphical Requirements Representation Method o Design-based Functional Testing	o Reviews o Database Management System (DBMS) o Design-based Functional Testing	o Compilers o Database Management System o Operating System o Reviews
ACTIVITIES	o Initial Requirements Review o Requirements Analysis o V,V&T Planning o Initial Test Case Generation o Interaction with customer o Sign-off by customer	o Refinement of Graphical Representation o Specify Information Design o Design Program Architecure & Allocate Requirements o Design Basic Control Flow o Test Case Generation o Preliminary Design Review	o Detailed Database Design o Detailed Module Desgn o Test Case Generation o Critical Design Review	o Code Devolopment o Module Testing o Function Testing o Acceptance Twsting

5.2.1 Requirements Subphase Activity Descriptions

5.2.1.1. Initial Requirements Review

The informal prose requirements for the AIMS is given in Figure 5.2-1. Appropriate management and technical personnel from the software development group review these requirements for completeness, consistency, and correctness and prepares a list of questions which address particular aspects of the requirements. This list is then supplied to the customer and a Requirements Review meeting is scheduled and held with customer and user, e.g. clerks, agents. During the meeting the questions are discussed with the goal being a more specific and unambiguous set of requirements.

V,V&T Purpose: To produce a requirements specification providing the foundation from which more formal requirements specification, V,V&T planning, and test planning will be accomplished.

V,V&T Technique: The review itself is the V,V&T technique used in this activity. Some of the questions addressed during the review could be:

o Shouldn't a claims record contain some kind of indication as to the nature of the claim? For example: if it is due to an accident, who was at fault?

o How is the "reasonableness" of a claim amount determined?

o How does one know what claim numbers are valid for which agents?

o When is the premium rate computed? How is it computed?

o Shouldn't the acceptance criteria include provisions for testing more than just the functional capabilities?

5.2.1.2. Requirements Analysis

The requirements analysis involves translation of the informal prose requirements into a formal representation. This will result in the identification of other aspects of the requirements which will need clarification or further definition. For this example, the graphical representation(GR) scheme used is a modification of the Systematic Activity Modeling Method [SAMM].

V,V&T Purpose: To identify inadequately specified requirements such as incomplete, ambiguous, or otherwise unclear requirements statements.

V,V&T Technique: Formal reviews serve as the V,V&T technique used to achieve the above purpose on this project. Problem issues which are identified during the requirements analysis are documented and distributed to the customer and a second Requirements Review is scheduled. This review again involves dialogue between the customer and the developers; it centers on the formal requirements statement and the identified issues. The result is a revised set of requirements in both formal and informal forms and a graphical

AIMS Requirements

A system called the Auto Insurance Management System (AIMS) is to be developed which will provide an automated set of automobile insurance support capabilities integrated through the use of a common database. The basic capabilities to be provided include: accounts payable, accounts receivable, management report generation, and database management.

System Information: Information contained in the database includes client records, claims records and the payout account. There is one client record for each policy holder and contains:

- policy number;
- name and address of the client;
- agent number;
- policy effective date and expiration date;
- name, birth date, sex, marital status, driving record of each driver;
- vehicle information - make, model, style, year;
- insurance coverage - comprehensive, medical, collision and deductable, premium rate classification, balance due, date due, credited amount (e.g., from prepaid premium), number of claims made on this policy and total amount paid out.

There is one claims record for each claim and contains:

- claim number and date of claim;
- associated accident report number;
- driver's name;
- payee (e.g., repair shop), name and address;
- agent number and policy number of client making claim.

There is one payout account record in the database and contains:

- account balance;
- date and time of the last change to the account balance;
- minimum allowed balance, date and time of last minimum change;
- maximum allowed balance, date and time of last maximum change;
- total year-to-date claims and premium total;

Accounts Payable: The accounts payable function processes claims transactions and issues payment checks to the payee which will either be the repair shop or the client. The transactions are input to the AIMS system from a file containing the day's claims. Having been input, a claims transaction is validated by checking the consistency of the client information contained in the transaction with that contained in the client record, by verifying that the claim number is valid for this agent, and by checking the "reasonableness" of the amount. Once validated, the new claims record is entered into the database, the amount of the claim is withdrawn from the payout account and the check is issued (i.e., printed). When the claims withdrawal is made from the payout account the account balance is updated and then compared to the minimum allowable balance. If the new balance is less, then a notice is issued (for management) indicating the situation. The date and time of last change to the account balance is also updated as well as the year-to-date claims total.

Accounts Receivable: The accounts receivable function issues policy notices and processes premium payments. Policy notices are issued by a batch program which runs once per day. The program reads all client records and checks each record to determine if a premium due notice or a cancellation (i.e., past due) notice should be issued and if so, prints the appropriate notice. When a premium payment is received a transaction is entered into a file which is processed daily by the AIMS to update client records with the premium payment. The information included in the transaction includes the policy number, client's name, agent number, due date and amount paid. Once received, the transaction is validated by verifying that the input data is consistent with that in the client record and that the amount paid is sufficient. The client record is then credited with the payment and the amount is deposited in the payout account. The payout account balance, year-to-date premium total, and date and time of last change to the account balance are updated when the deposit is made. Note that the maximum allowable balance is not automatically checked. This is a manual function performed by management.

Management Reports: Four management reports are produced: claims report, new clients report, client profile report and client cancellation report. The claims report is produced on a monthly basis and gives the total number of claims and the total amount paid in claims for a given month. It also performs a trend analysis based on totals from prior months. The new clients report is produced on a monthly basis and lists new clients and their coverage for a given month. The clients are grouped by agent which allows management to view the sales progress of each agent. A company-wide sales trend analysis is also produced. The client profile report provides accident statistics based on driver's age, sex, marital status, etc. This report is produced semi-annually. The client cancellation report lists the clients which were cancelled during a given month and the reason for the cancellation.

Database Management: Database management activities provide for client and payout account management capabilities. Client records are entered, queried, modified and cancelled (i.e., made inactive but not deleted). A log of all client transactions is also stored in the database (or possibly on tape). Payout account functions include query, modification of the balance limits, external deposits and external withdrawals. External deposts and withdrawals are made from/to other company financial resources whenever the account balance exceeds the allowable limits. A log of all deposits and withdrawals (including premiums and claims) is kept on the database.

Acceptance Criteria: The acceptance criteria for the AIMS is the successful execution of a set of acceptance tests. An execution is successful if it correctly performs the desired function within the required execution time. The acceptance tests are specified by an independent team and reviewed by management, user and development personnel.

Figure 5.2-1 Informal Prose Requirements

MICROFILMED FROM
BEST AVAILABLE COPY

representation(GR). Specific activities which are performed within this review are:

- o Verification that all requirements have been correctly represented using the formal scheme,
- o Identification of the problems encountered during the restatement elaboration of the requirements, and
- o Discussion and resolution of the problems.

Example:

The formal representation for the basic system and the accounts payable function are shown on the following pages(figure 5.2-2). The graphical representation is interpreted as follows:

Master input files are at the top of the diagram and are represented by numbers.

Master output files are at the right of the diagram and are represented by numbers.

Files internal to a data flow diagram are labeled with a letter followed by a number.

The upperhalf of figure 5.2-2 is the root which contains five modules, A-E. The data flow within the root and to and from master files are labeled according to their source. If the data are internal to the root, its identifier is preceded by the module letter.

The lower half of figure 5.2-2 is an expansion of module A from the root. The lower left corner of each box contains the parent, i.e. A in the root. The lower right corner of each box is the letter designator for each module, i.e. A-E. Data created by the accounts payable activity is labeled according to its source, e.g. data B.1, a validated claims transaction, is created by module B, validate claims transaction, and used by modules C-E. Data B.2, invalid claims transaction notice, is created by B and put on master file 7, user/client notices.

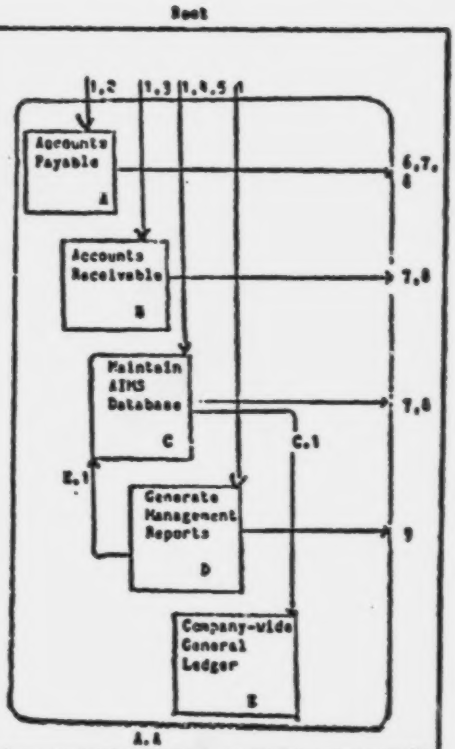
Some of the problems which could be identified are:

- o What does the system do with an invalid claims transaction? Solution: Output a notice to the user identifying the errors.
- o The involved driver's record in the client's record needs to be updated to reflect a new claim due to an accident. There does not appear to be enough information in the client record for this. Solution: Add the necessary information to the claims transaction.

ACTIVITY - DATA FLOW DIAGRAM

title
Auto Insurance Management System (AIMS)

DATA ID	DATA DESCRIPTION
1	AIMS data base
2	Claims transaction file
3	Premium payment transaction file
4	Client transaction
5	Payout account transaction
6	Claim payment check
7	User/client notices
8	Updated AIMS data base
9	Management reports
C.1	AIMS account information
E.1	AIMS account transactions


Root
1,2
1,3
1,4,5
Accounts Payable
A
6,7,8
Accounts Receivable
B
7,8
Maintain AIMS Database
C
7,8
C.1
E.1
Generate Management Reports
D
9
Company-wide General Ledger
E
A.A

ACTIVITY - DATA FLOW DIAGRAM

title
Accounts Payable

DATA ID	DATA DESCRIPTION
1	AIMS database
2	Claims transaction file
A.1	Client record
B.1	Validated claims transaction
B.2	Invalid claims transaction notice
C.1	Claims record
D.1	Claims payment check
D.2	Updated payout account
D.3	Insufficient funds notice
D.4	Claims transaction log
E.1	Updated client record
E.2	Cancellation notice
E.3	Rate increase notice

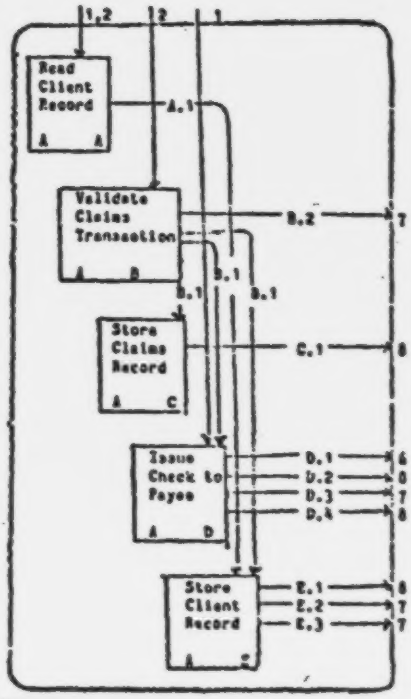

1,2
2
1
Read Client Record
A A
A.1
Validate Claims Transaction
A B
B.2
7
B.1
B.1
B.1
Store Claims Record
A C
C.1
8
Issue Check to Payee
A D
D.1
6
D.2
8
D.3
7
D.4
8
Store Client Record
A E
E.1
8
E.2
7
E.3
7

Figure 5.2-2 Requirements Graphical Representation

MICROFILM
BEST AVAIL

5.2.1.3. V,V&T Planning

V,V&T planning is one aspect of the overall planning process. It is accomplished in parallel with other planning activities and the requirements identification activity.

V,V&T Purpose: To choose V,V&T practices which can be implemented to suit the project needs. The objectives are:

- o Identify the goals of the AIMS project's V,V&T activities,
- o Select supporting V,V&T techniques and tools, and
- o Develop plans for each phase's V,V&T activities. (Plans include tasks, e.g., acquiring or developing tools), schedules, responsibilities, and resources.)

V,V&T Technique: The process described in Chapter 4 is used to develop the V,V&T plans.

5.2.1.4. Initial Test Case Generation

The AIMS requirements will be analyzed and test cases will be designed to test the functional capabilities of the system. These test cases will also form the basic set of acceptance tests.

V,V&T Purpose: To design test cases which, when used to test the AIMS software, will maximize the possibility of revealing the presence of errors in the software.

V,V&T Technique: Requirements-based functional testing is applied to generate this initial set of test cases.

Example:

> In the accounts payable function a claims transaction is validated by checking (among other things) that the claim number is valid for the given agent. Each agent has a specified range in which claim numbers associated with claims issued by that agent must fall. Assuming an agent was assigned claim numbers in the range 801000 to 801999, test cases which are generated to test accounts payable should include claim numbers as follows.

Test Data Class	Test Claim Number	Expected Output	Comment
Non-extremal	801500	None	valid
Non-extremal	801317	None	valid
Extremal	801000	None	upper bound
Extremal	801999	None	lower bound
Extremal	800999	Invalid Claim Number	lower bound-1
Extremal	802000	Invalid Claim Number	upper bound+1
Special	80100A	Invalid Claim Number	
Special	80100Z	Invalid Claim Number	
Special	80150	Invalid Claim Number	
Special	-01500	Invalid Claim Number	
Special	80L500	Invalid Claim Number	

5.2.2 Preliminary Design Subphase Activity Descriptions

5.2.2.1. Refinement of Graphical Representation

The GR diagrams developed during requirements analysis will be decomposed to reflect the requirements for the system in more detail.

V,V&T Purpose: The completeness and consistency of the GR description of the requirements and preliminary design should be ensured.

V,V&T Technique: A review of the resulting diagrams will be performed to verify:

- o that all of the basic activities necessary to perform a particular function have been identified,
- o that all inputs and outputs required by each activity have been identified, and
- o the consistency and completeness of the data flows.

Example:

> Within the accounts payable function there is no indication as to the action which is to be taken when a claim transaction is processed for a claim which has been previously entered. This error would be discovered during the review of the GR activity for the accounts payable function.

5.2.2.2. Specify Information Design

The preliminary design of the information consists of a detailed user input/output specification and a description of the basic content and structure of the data used by the system. The detailed user input/output specification essentially amounts to preparing a user's manual for the system. The formats used to input claims and premium payment transactions are defined as well as the output responses. The printed report formats for the management reports are also defined. Specification of the basic data structures and content will consist of identification of variables and records needed by the system, and the relationships which exist among them.

V,V&T Purpose: The V,V&T purpose in this activity is twofold. First, the detailed user specifications need to be shown to be usable and that they satisfy the needs of the user. Second, the system data structures and content need to be verified and shown to be complete (i.e., that which is required to perform all system functions) and correct (i.e., the data types and relationships are consistent with the functions which need to be performed).

V,V&T Technique:

- o A formal review will be held with the customer to review the detailed user input/output specifications. This will be preceded by informal dialogue between the user community and the developers to aid in the development of the specifications. Once satisfied, the customer will formally sign-off on the specification.

- Formal inspections of the system data structures and content will be performed.

Example:

Discovered by the customer participating in the formal review of the detailed input/output spec was that a client is not always the owner of the car, so that lien-holder information needs to be included in the client record.

5.2.2.3. Design Program Architecture & Allocate Requirements

The program architecture design gives a complete high-level description of the software. It refines and groups functions defining software components and interfaces.

V,V&T Purpose: Requirements are cross-referenced by the design to ensure that all the requirements have been addressed.

V,V&T Technique: Specification tracing.

Example:

A complete set of cross-references is defined and maintained. These show the evolution from the prose requirements to the requirements represented by the GR and finally to the components identified in the design.

5.2.2.4. Design Basic Control Flow

The GR represents the data flow within a system but only shows control flow in an implicit way. The system's control flow therefore needs to be explicitly designed. The activities identified in the GR will need to be mapped into modules. The control flow between modules must also be described. This is done using an informal design language. This defines the program architecture. The hierarchical structure of the modules comprising the system are developed.

V,V&T Purpose: To produce a correct and understandable description of the basic control flow of the system.

V,V&T Technique: An inspection of the control flow design will be performed to verify:

- its consistency with the GR representation,
- correctness of the high level logic, and
- the quality of the modularization, i.e., are the functional boundaries natural?

5.2.2.5. Test Case Generation

V,V&T Purpose: Generate test data that will exercise and test each function, and also demonstrate the code is consistent with the design.

V,V&T Technique: Design-based functional testing.

Example:

> Test cases for a function which adds the amount of the premium payment to the payout account would include: a negative (or zero) amount, an amount which is greater than zero but less than that which would leave the balance larger than the maximum allowed, and one which would leave the balance greater than the maximum allowed.

5.2.2.6. Preliminary Design Review

At the completion of the preliminary design activity, a formal review is held. This involves management and technical staff representing the developer and the customer/user. It covers all aspects of the design. Results of V,V&T activities are reviewed. Management of customer/user and developer sign-off of acceptance is required.

5.2.3 Detailed Design Subphase Activity Descriptions

5.2.3.1. Detailed Database Design

The format and structure of the data to be stored in the system database is designed. This includes description of data which are logically related in the form of records, and the relationships which exist between records. The logical structure of the database will be described using a graphical database design representation. Record descriptions will be specified in a Data Definition Language. Examples are shown in figures 5.2-3 and 5.2-4.

In figure 5.2-3, ovals represent record access (key) fields, boxes represent records, "1:M" means that for each client record there are potentially many (1 or more) claims records.

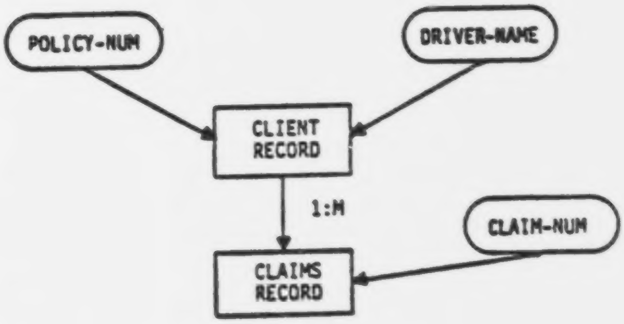


Figure 5.2-3 Sample Database Schema Showing Client-Claims Relation

```
record name is CLAIMS
 location mode is calc in CALC-KEY using POLICY-NUM
  01   CLAIM-NUM                              PIC 9 (6)
  01   DATE-OF-CLAIM                          PIC 9 (6)
  01   ACC-REP-NUM                            PIC 9 (9)
  01   DRIVER
       02 LAST                                PIC X (15)
       02 FIRST                               PIC X (15)
       02 MIDDLE-INTL                         PIC X
  01   PAYEE
       02 NAME                                PIC X (31)
       02 ADDRESS
          03 STREET                           PIC X (24)
          03 CITY                             PIC X (15)
          03 STATE                            PIC X (2)
          03 ZIP                              PIC X (5)
  01   POLICY-NUM                             PIC 9 (8)
  01   AGENT                                  PIC 9 (5)
```

Figure 5.2-4 Sample CLAIMS Record Description

V,V&T Purpose: The database design needs to be verified for consistency with the preliminary design. Moreover, the database structure will be verified to ensure that it is correct and is reasonable with respect to potential storage consumption and access time.

V,V&T Technique: An inspection of the database design is performed to ensure that the above V,V&T purpose is met.

Example:

During the inspection of the database design an error is found in the claims record (Figure 5.2-4) where POLICY-NUM is identified as the key field where as the schema diagram (Figure 5.2-3) indicates CLAIM-NUM. The solution is to change the key field in the claims record description to CLAIM-NUM.

5.2.3.2. Detailed Module Design

Detailed module design includes for each module a description of the function performed and descriptions of input and output data, as well as a high level description of how the function is to be done (i.e., the algorithm used).

V,V&T Purpose: To show that 1. all of the system's functional capabilities are addressed by one or more modules and 2. each module addresses one or more system functions. Moreover, relationships among and interfaces between all modules are identified and verified.

V,V&T Technique:

- Inspections of the system modules are held. Activities performed during the inspections include manual checking of the module interfaces to ensure that all modules are used and that their inputs and outputs are consistent. Inspections are also used to verify informally the correctness of the algorithms used.

- Requirements tracing is accomplished by identifying each module with the lowest level GR activity (from the preliminary design) in which the module is contained.

Example:

A module which updates the date and time of the last access to the payout account record has as one of its inputs the premium payment transaction. However, manual interface checking detects an inconsistency whereby the premium payment transaction is not supplied. As it turns out, the transaction is not used within the module and is deleted as an input.

5.2.3.3. Test Case Generation

This involves refining and adding to test data previously developed.

V,V&T Purpose: Test cases are developed to exercise and test the internal structures and functions of modules.

V,V&T Technique:

- Branch testing
- Path testing

Example:

The module which validates a claim number checks for six error conditions. Associated with these conditions are three actions. Test data is developed to exercise all combinations of error conditions and resulting actions, that is all branches and all paths through the modules.

5.2.3.4. Critical Design Review (CDR)

At the completion of the detailed design a formal detailed design review is held. This primarily involves project management and technical personnel and covers all aspects of the design (including the test cases). Management sign-off of their acceptance of the design is required.

5.2.4 Programming Subphase Activity Descriptions

5.2.4.1. Code Development

The detailed design of a given component provides the information needed to write the code for that component in the host programming language ,e.g., COBOL. Once written, the code is entered into the computer and all compilation errors are removed.

V,V&T Purpose: V,V&T of the compiled code is performed to:

- Verify the consistency of the code with the detailed design,
- Identify errors, and
- Ensure adherence to programming standards.

V,V&T Technique: Formal reviews of each system module.

Example:

During an inspection of "issue policy notices" module the section of code which is responsible for issuing a premium due notice is found to be in error. The error is that the premium due notice is printed without having the appropriate data moved into the printer buffer. A sample portion of the inspection checklist used is shown below in Figure 5.2-5. This particular error is discovered using question two under "data reference."

DATA DECLARATION

1. Are all variables declared?
2. Are the correct attributes assigned?
3. Are variables properly initialized?
4. Are variable naming conventions followed?
5. Is the proper explanatory comment included for each variable?

DATA REFERENCE

1. Are there any unreferenced variables?
2. Are there any references to unassigned variables?
3. Are subscripts within range?
4. Are there off-by-one errors in subscript computations?

⋮

Figure 5.2-5 Sample Portion of Code Inspection Checklist

5.2.4.2. Module Testing

An incremental, bottom-up testing strategy is used to test the AIMS modules. This involves individually testing the lowest level modules; then combining and testing those modules with the higher level modules which call them. The process continues until all modules are combined into the complete system. Test drivers are written to control the testing of the individual modules. The test data used is that created by design-based functional testing which were generated from analyses of the functional, structural and interface specifications of the individual modules during detailed design.

V,V&T Purpose: To reveal errors present in the individual modules.

5.2.4.3. Function Testing

Function testing of AIMS uses the test cases developed from requirements-based functional testing during preliminary design to test the functional capabilities of the AIMS software.

V,V&T Purpose: To reveal errors where the software fails to perform a function as specified in the requirements.

5.2.4.4. Acceptance Testing

Acceptance testing is similar to function testing in that it consists of a subset of the same test cases. But where the purpose of function testing is to reveal the presence of errors, acceptance testing is to demonstrate that the software performs according to its specification. Acceptance testing is a formal procedure and requires customer sign-off.

5.3 EXAMPLE 2: SOFTWARE DEVELOPMENT USING A MINIMALLY AUTOMATED V,V&T TOOL SET

The minimally automated V,V&T tool set consists of a set of commonly available tools. These are used to supplement the manual techniques described earlier. The tools contained in the minimal set and the lifecycle phase in which they are applied are shown below.

TABLE 5.3-1
EXAMPLE 2 SUMMARY
SOFTWARE DEVELOPMENT USING A MINIMALLY AUTOMATED V,V&T TOOL SET

SUBPHASES	REQUIREMENTS	PRELIMINARY DESIGN	DETAILED DESIGN	PROGRAMMING
INPUT	• (No Additional Input)	• Detailed Requirements Specification Including Use of Requirements Specification Language	• Preliminary Design Document Including Use of Requirement and Design Specification Languages	• Detailed Design Document Including Use of Design Specification Language
OUTPUT	• Detailed Requirements Specification Including Use of Requirements Specification Language	• Preliminary Design Document Including Further Use of Requirements Specification Design Language	• Detailed Design Document Including Use of Requirement and Design Specification Languages	• (No New Outputs)
SUPPORTING TECHNOLOGY	• Requirements Specification Language • Requirements Specification Language Analyzer	• Requirements Specification Language • Tool-Supported Design Language	• Design Specification Language	• Cross-Referencer • Test Coverage Analyzer • File Comparator
ACTIVITIES	• Requirements Analysis	• Specify Information Design • Design Basic Control Flow	• Detailed Module Design	• Code Development • Module Testing • Function Testing

- o Requirements
  - Requirements representation language analyzer

- o Preliminary and Detailed Design

  - Design representation language analyzer

- o Code
  - Cross reference generator
  - File comparator
  - Test coverage analyzer

5.3.1 Requirements Subphase Activity Description

A tool-supported requirements statement language is used to formally specify the requirements. It is used in conjunction with the graphical represention technique which analyzes the gross requirements while the language representation is used to perform a more detailed analysis. It is used to directly support the V,V&T purpose to identify inadequately specified requirements in that a consistency analyzer is part of the tool. The reports generated by the tool are included with the other material for requirements reviews.

Example:

A Problem Statement Language (PSL) [TEIC] description of the accounts payable function is shown in Figure 5.3-1.

```
/* Auto Insurance Management Systems (AIMS)
   Accounts payable */

INPUT:  claims-transaction-file;
OUTPUT: claim-payment-check, client-notices;
SET:  aims-data-base
INTERFACE:  client;
   GENERATES:  claims-transactions;
   RECEIVES:  claim-payment-checks;
PROCESS:  accounts-payable;
   UPDATES:  aims-data-base;
   RECEIVES:  claims-transactions;
   GENERATES:  claim-payment-checks;
EOF
```

Figure 5.3-1 PSL Specification for Accounts Payable

In this example, the Problem Statement Analyzer (PSA)[TEIC] would identify an inconsistency with respect to the accounts payable process. Accounts payable generates claim-payment-checks which have not been previously defined (it is misspelled in the OUTPUT definition).

5.3.2 Preliminary Design Subphase Activity Descriptions

5.3.2.1. Specify Information Design

A formal requirements specification language (such as PSL) can be used to specify a high-level system information design. Reports produced by the tool are included in the preliminary design document and are analyzed during the reviews.

5.3.2.2. Design Basic Control Flow

One of the difficulties encountered in the use of an informal design language is that of ensuring a consistent representation for all modules. The use of a tool-supported design language, such as Program Design Language (PDL) [CAIN], prevents this from happening.

Example:

Figure 5.3-2 shows the PDL description for "Issue Policy Notices". The PDL for all modules is included in the reviews and walk-throughs.

```
Issue Policy Notices

      Initialize
      Fetch first client record
      do until end-of-file on client file
                if premium is due
                  Issue premium due notice
                else if premium past due
                  Issue cancellation notice
                endif
                Fetch next client record
       enddo
```

Figure 5.3-2 Preliminary Design PDL of "Issue Policy Notices" Function

5.3.3 Detailed Design Subphase Activity Description

As in the preliminary design described above, a formal, tool-supported design language is used to describe modules and their algorithms. The reports generated by the tool are included with the information used in the reviews and walkthroughs.

During the inspection of this module two problems, an error and a standards violation, are identified. The error is that the program may very likely issue more than one premium notice. The problem occurs in line six. If current-date is greater than or equal to premium-due-date - 40 then current-date +1 (when the program is next executed) will also be greater. One solution is to introduce an additional boolean variable in the client record, "issued", which is initially false but set to true when the notice is issued. Line six then becomes:

```
if(current-date >=  premium-due-date - 40) and not issued
```

The standards violation is a result of the use of the integer literal constant "40" in line six, particularly since this value may be subject to change. A program constant called "response-interval" should be defined with the value 40 and referenced in line six.

Example:

Figure 5.3-3 shows the PDL of the detailed design for "Issue Policy Notices" from the preliminary design shown in Figure 5.3-2.

```
Issue Policy Notices

    Get current-date
    Issue operator instructions
    Open data base
    Fetch first client record
    do until end-of-file on client file
        if current-date > = premium-due-date - 40
            Issue premium notice
        else if current-date ≠ premium-due-date
            Issue cancellation notice
        endif
        Fetch next client record
    enddo
    Close database

Issue operator instructions

Display program start message and date
Display prepare printer message
Wait until printer ready
```

Figure 5.3-3 Detailed Design PDL of "Issue Policy Notices" function

5.3.4 Programming Subphase Activity Description

5.3.4.1. Code Development

In addition to that described earlier, a cross-referencer is used to produce cross-reference lists of all identifiers used by a program. This list is included with the source code listings for module inspections.

Example:

A careful examination of the cross-reference listing of module ISSUE-CHECK in ACCOUNTS-PAYABLE during the code inspection indicated that two variables, PAYOUT-ACCOUNT-BAL and PAYOUT-ACCT-BAL, were referenced. The error was that PAYOUT-ACCOUNT-BAL, should have been coded "PAYOUT-ACCT-BAL."

5.3.4.2. Module Testing

A test coverage analyzer is used to supplement module testing as described in Table 5.3-1. Each module to be tested is instrumented to collect execution frequency counts and then executed. The execution counts for each statement are then listed with the corresponding statement by a post-execution routine. Untested or poorly tested portions of the module can be identified and additional test cases can be generated to test those specific segments.

Example:

> ACCOUNTS-PAYABLE processes claims transactions read from a file which contains a given day's claims. The module contains a check to verify that each record is indeed a claims transaction and, if not, invokes an error handling routine which logs the error. Use of a test coverage analyzer showed that this particular situation did not arise during testing of the module using the tests created during detailed design. As a result, those tests are supplemented with invalid claims transactions and the module retested. This, in turn, results in an error being revealed whereby the error handler responds with an incorrect output response.

5.3.4.3. Function Testing

Function testing is supplemented with the use of a file comparator. Associated with each of the requirements-based functional test cases is the expected output. This is stored on a file in the exact format expected to be produced. When the AIMS software is tested, the resulting output is stored on a separate file. A file comparator is used to detect automatically any discrepancies which may have occurred.

Example:

> In preparing the test cases for the New Clients report, a form is used which formats the expected output data in accordance with the specification. Each report corresponding to a given test case is then stored on a file in the order in which the tests are to be executed. Testing is then performed and the actual output is compared to the expected output using a file comparator. The results show the presence of two errors, a format error and a data output error. The format error is a misalignment caused by incorrect spacing between output fields. The data output error is a missing agent name which is to be printed with the agent number.

5.4 EXAMPLE 3: SOFTWARE DEVELOPMENT USING A FULLY AUTOMATED V,V&T TOOL SET

The fully automated V,V&T tool set is comprised of commonly used tools and techniques, and includes those tools contained in the minimal tool set described earlier as well as those described in this section. The additional tools and the applicable lifecycle phase are shown below.

- Preliminary Design
  - Assertion generation
- Detailed Design
  - Assertion generation
- Code
  - Interface checker
  - Data flow analyzer
  - Assertion processor
  - Standards analyzer

TABLE 5.4-1
EXAMPLE 3 SUMMARY
SOFTWARE DEVELOPMENT USING A FULLY AUTOMATED V,V&T TOOL SET

SUBPHASES	REQUIREMENTS	PRELIMINARY DESIGN	DETAILED DESIGN	PROGRAMMING
INPUT	• (No Additions to Minimal Tool Set)	• (No Additional Inputs)	• Preliminary Design Document Including Assertions	• Detailed Design Document Including Assertions
OUTPUT	• (No Additions to Minimal Tool Set)	• Preliminary Design Document Including Assertions about the Design	• Detailed Design Document Including Additional Assertions	• (No Additional Outputs)
SUPPORTING TECHNOLOGY	• (No Additions to Minimal Tool Set)	• Assertion Generation	• Assertion Generation	• Interface Checker • Data Flow Analyzer • Assertion Processor • Standards Analyzer
ACTIVITIES	• (No Additions to Minimal Tool Set)	• Design Basic Control Flow	• Detailed Module Design	• Code Development • Module Testing

5.4.1 Requirements Subphase Activity Description

(No additions to minimal tool set.)

5.4.2 Preliminary Design Subphase Activity Description

V,V&T Technique: Assertion generation is used to specify the desired functional properties of the individual modules. This is done by including in the module specifications input and, to the extent possible, output assertions.

Example:

Policy numbers are stored in the database in blocks of arrays where each block contains a fixed number (n) of policy numbers (policy-num) and the address (policy-addr) of their associated client records. Policy numbers are stored in the policy-num array in ascending order. A procedure, find-policy, is called to search the policy-num array for a supplied policy number and return the address of its client record. If the supplied policy number is not found an address of zero is returned. The input and output assertions which capture the functional properties of find-policy are given below.

```
1) /* assert input policy-num (1)< = num< = policy-num (m) */ and
2) /* assert input forall i in 1...n-1:polcy-num (i)< = policy-num (i+1) */
3) /* assert output (exists in i in 1...n : num = policy-num (i)) */ or
4) /* assert output (addr=0 and forall i in 1...n:num = policy-num (i)) */
```

5.4.3 Detailed Design Subphase Activity Description

V,V&T Technique: Assertions are generated to include algorithmic detail in addition to input and output specifications of the functional properties of the individual modules.

Example:

The example in the previous section describes the find-policy procedure and specifies the input and output assertions associated with it. Shown in figure 5.4-1 is the PDL for find-policy which is implemented using a binary search algorithm.

The input and output assertions capture the functional properties of the procedure independent of the algorithm used to implement the search. Assertions 1, 2 and 3, however, capture conditions which are very dependent upon the algorithm. Assertion 1 is always correct whenever num is in the policy-num array. If num is not in the array, assertion 1 is violated the last time through the loop (when high = low). This is an acceptable result, however, in that num should be a valid policy number.

```
find-policy:

  /* searches sorted global array policy-num for num (input argument) and, if
   found, returns the associated policy-addr in addr (output argument). If
   not found a zero is returned in addr */

  /* assert input policy-num (1)< = num< = policy-num (n) */
  /* assert input forall i in 1...n-1: policy-num (i)< = policy-num (i+1) */

  set addr to 0
  set low to 1
  set high to n
  do until high< low or num = policy-num (i)
(1) /* assert 1< = low< = high< = n and policy-num (low)< = num<=
        policy-num (high) */
   set m to (low + high) /2
   if num< policy-num (i)
   set high to m-i
   else if num> policy-num (i)
   set low to m+i
   else goto successful
  enddo

 /* unsuccessful */

(2) /* assert high = low-1 and policy-num (high)< num< policy-num (low) */
    /* assert output addr = 0 and forall i in 1...n: num ≠ policy-num (i) */
  return

 /*successful*/

  set addr to policy-num (i)
(3) /* assert 1< = low< = m< = high< = n and num = policy-num (m) */
    /* assert output exists i in 1...n: num = policy-num (i) */
  return

 end find-policy;
```

Figure 5.4-1 Detailed PDL with ASSERTIONS

5.4.4 Programming Subphase Activity Descriptions

5.4.4.1. Code Development

The code development activities described in earlier sections are supplemented in a full tool set environment with an interface checker, data flow analyzer and standards analyzer. These tools can be separate but are often included as capabilities provided by a single tool. They are all static analysis techniques and are therefore applied prior to software testing. The output which results from each of the capabilities is included with the material for the formal code inspections.

V,V&T Techniques:

- Interface checking is used to check the consistency of the interfaces between modules.

Example:

An error is detected between the module which reads client records for premium payment processing and the "find-policy" module. It is an inconsistency in the type of the arguments for the policy numbers. "Find-policy" is being called with a policy number of type character where it should be type integer.

- Data flow analysis is used to identify variable reference/definition anomalies.

Example:

When data flow analysis is performed on the module which updates the payout account with a premium payment, a reference to an uninitialized variable is noted. The variable should contain the current date and time and is used to update the date and time of the last change to the payout account. A call to the routine which updates the time and date should be made prior to the reference.

- Standards' analyzers are used to assure adherence to program coding and documentation standards. One of the primary capabilities provided by most commonly available standards' analyzers is the notification of the use of non-standard language features.

Example:

One of the requirements for the AIMS software is that it be portable. To assist in the development of portable code, a COBOL standards' analyzer is used. All places where a standards' violation occurs is either changed or justified. Even trivial nonstandard features such as the use of the abbreviation "DISP" for "DISPLAY" are detected. In addition, a variety of undesirable standard language constructs such as the "ALTER" statement and "NEXT SENTENCE" clause are detected with the tool.

5.4.4.2. Module Testing

The module testing activities described in earlier sections are supplemented in a full tool set environment with a dynamic assertions processor. This processor is generally included as part of a broader dynamic analysis tool which includes, for example, statement execution counts.

V,V&T Technique: Assertions processor: A dynamic assertions processor translates assertions, usually specified as part of the source program, into source language statements which check the validity of the assertion during program execution. Generally, when an assertion is violated, an informative message is output.

Example:

Figure 5.4-2 shows a portion of a FORTRAN implementation of the find-policy routine from Figure 5.4-1. Also shown is an example of an assertion violation message which was printed when the assertion in line 14 of the program was violated (i.e., false) during program execution. Subsequent analysis of the problem indicated that the error was an incorrect coding of line 18 from the PDL where HIGH should have been set to M - 1, not M + 1.

```
                .
                .
                .
13     100  CONTINUE
14 C*           ASSERT(1.LE.LOW.AND.LOW.LE.HIGH.AND.HIGH.LE.N
15 C*           .AND.POLNUM(LOW).LE.NUM.AND.NUM.LE.POLNUM(HIGH))
16              M = (LOW + HIGH)/2
17              IF ( NUM .LT. POLNUM(M) ) THEN
18                  HIGH = M + 1
19              ELSE IF ( NUM .GT. POLNUM(M) ) THEN
20                  LOW = M + 1
21              ELSE
22                  GO TO 200
23              ENDIF
24              IF(HIGH.LE.LOW.AND.NUM.NE.POLNUM(M)) GO TO 100

                .
                .
                .

*** ASSERTION VIOLATION AT LINE 14 OF SUBROUTINE FNDPOL:
    CURRENT EXECUTION COUNT = 2
    LOW = 1, HIGH = 65, N = 64, NUM = 22707,
    POLNUM(LOW) = 16747, POLNUM(HIGH) = 36757
```

Figure 5.4-2
Find-policy Subroutine and Corresponding Assertion Violation Message

5.5 EXAMPLE 4: SOFTWARE MAINTENANCE

System maintenance activities involve the processing of changes to the system software and documentation necessary to correct an error or to enhance or alter the system's functional capabilities. The maintenance activities described in this example are supported by the fully automated V,V&T tool set, described in earlier examples, augmented with software configuration control procedures.

Maintenance is generally divided into two classifications:

- o Problem reporting and correction, and
- o Change request processing.

Problem reporting and correction involves a formal notification (usually consisting of a form which is filled out by a user) of an error in the software. The error is then verified and the offending module(s) identified and corrected. The effort required for the entire process is, in most cases, minimal in that any more than a very minimal redesign is seldom necessary.

Change request processing involves a formal notification (basically consisting of a requirements statement) of a desired enhancement, alteration or improvement (e.g., performance) in the system's functional capabilities. Change requests always require some software redesign and, as a result, involve a greater level of effort.

Each of these activities are illustrated in the following sections with some simple examples from the AIMS software.

TABLE 5.5-1
EXAMPLE 4 SUMMARY
SOFTWARE MAINTENANCE

SUBPHASES	PROBLEM REPORTING AND CORRECTION	CHANGE REQUEST PROCESSING
INPUT	o System Software o System Documentation o System Test Cases o Problem Reports	o System Software o System Documentation o System Test Cases o Change Requests
OUTPUT	o Updated System Software o updated System Documentation	o Updated System Software o Updated System Documentation
SUPPORTING TECHNOLOGY	o Fully Automated V,V&T Tool Set o Configuration Control Procedures	o Fully Automated V,V&T Tool Set o Configuration Control Procedures
ACTIVITIES	o Problem Analysis o Problem Correction	o Requirements Analysis o Redesign and Coding

5.5.1 Problem Reporting and Correction Activity Descriptions

5.5.1.1. Problem Analysis

When a problem report is received, the problem must be analyzed to determine the cause of the error. Often, the error needs to be recreated to obtain additional information pertaining to the source of the error. It may be that a user has made an error which was the result of a misunderstanding on his part. The misunderstanding could be the user's problem or it may be the user documentation that is incorrect.

If the problem is indeed an error in the software, then the offending code needs to be identified and possible solutions addressed. This is generally not an easy task. If the software has been developed using the lifecycle V,V&T techniques and tools described in the previous sections, the error has escaped very thorough attempts to uncover it. Although some V,V&T tools can provide assistance, they are generally not very helpful in locating the error. The best tool to use is one's brain. Tools are not meant to replace thinking but to assist in amplifying the thinking process.

Once the error has been located, possible solutions can be identified. These solutions vary from being very trivial code modifications to an extensive redesign. It is important to note that if good lifecycle V,V&T practices are utilized during software development, then the instances of redesign are significantly reduced.

Example:

> An AIMS Problem Report is received describing an error in which a claims transaction is due to an invalid agent number although that particular agent number is listed in the user documentation as valid. Upon analysis of the problem, the cause of the error turns out to be that a new agent was added to the company with a new number assigned. The documentation was updated but the table of agent numbers in the AIMS software was not updated.

5.5.1.2. Problem Correction

When the source of an error is found to be in the software and the offending code is identified, the correction of the error needs to be made. The code in the module which contains the error is modified. The V,V&T techniques and tools which are used during the code correction process will depend on the extent of the error. In the previous example, only a table addition is required. The compiler detects most errors and an inspection of the compiled table is generally all that is necessary before retesting. In a case where an entire algorithm is wrong, a formal review of the code is probably necessary, as well as use of the full automated V,V&T tool set to completely retest the module.

After the code has been modified and module testing completed, regression testing is performed. In cases where the module interfaces with another module(s), interface checking, hierarchy, module calling, etc. tools should be employed before regression checking. All system components which can potentially be affected by the change are retested using a standard set of system test cases augmented with tests aimed at revealing the original error (e.g., the actual input which caused the error should be included). When testing is complete, the actual output is compared with the expected (correct) output, possibly using a file comparator. If no errors are present, then the updated modules are incorporated into the production software and any pertinent documentation is updated.

Example:

> An error was discovered where on midnight of December 30, 1980 the year-to-date premium total was reset so that premiums received on December 31 were counted in the 1981 total instead of the 1980 total. Upon examination, it was found that the end of year test was 365 days. 1980, however, being a leap year, contained 366 days and so the reset occurred a day early. In this case, the software correction is quite simple and is implemented with little difficulty. In fact, it is more difficult to correct the misinformation in the database than it is to correct the error.

5.5.2 Change Request Processing Activity Descriptions

5.5.2.1. Requirements Analysis

The requirements analysis associated with system enhancements or alterations is essentially the same as described earlier in this chapter except for one fundamental difference. For all practical purposes, the feasibility of a particular enhanced or altered capability depends upon how well that capability can be implemented within the existing design. The more redesign that is necessary, the greater the magnitude of the cost to implement it. Redesign costs are not an issue during the requirements phase of the initial software development since there is no existing design. What this means is that the scope of what is or is not feasible is much narrower during the maintenance phase and therefore must be given consideration.

The V,V&T techniques applied during this requirements analysis are the same as before utilizing reviews, representation aids, etc. to assist in the analysis and specification.

Example:

> High level company management issued a change request to enhance AIMS to interactively process claims and premium payment transactions. The current system batchs the transactions on a file which is then processed once a day. This results in a response time which is inconvenient for the agents as well as the customer. It is decided to enhance the system to process claims and premium payments interactively as they are received.
>
> During requirements analysis, it is found that those system modules which are impacted by the redesign are those which control the input/output of the batch transactions. Replacement of those modules with ones which could perform interactive processing is relatively straightforward. Increased computer usage would, however, require additional hardware resources for normal production operation.
>
> Management felt that the extra cost was worthwhile and thereby authorized the system change.

5.5.2.2. Redesign and Coding

The process followed for the implementation of a system enhancement or an alteration is no different than that followed for a complete software development as described earlier in this chapter. Therefore, no further discussion is required.

CHAPTER SIX

SUMMARY

This document has presented information on lifecycle V,V&T and guidance on planning for V,V&T. It has covered:

- Software development phases and accompanying V,V&T activities
- A categorization of V,V&T techniques into static, dynamic, and formal analysis
- Suggestions on how to integrate static, dynamic, and formal V,V&T analysis into a lifecycle
- Guidance on steps and approaches to take when planning V,V&T activities
- Recognition that V,V&T activities must be tailored to meet the needs of specific projects

A series of rather detailed examples of V,V&T approaches using increasing levels of automated support were also presented. A reference guide of techniques and tools [POWE] for V,V&T is a valuable companion for this document. A glossary of V,V&T terms is included as an appendix.

GLOSSARY

BLACK BOX TESTING: see FUNCTIONAL TESTING

BOUNDARY VALUE ANALYSIS: a selection technique in which test data is chosen to lie along "boundaries" or extremes of input domain (or output range) classes, data structures, procedure parameters, etc. Choices often include maximum, minimum, and trivial values or parameters. This technique is often called stress testing.

BRANCH TESTING: a test method satisfying coverage criteria that require, for each decision point, each possible branch be executed at least once.

CAUSE EFFECT GRAPHING: test data selection technique. The inputs and outputs of the program are determined through analysis of the requirements. A minimal set of inputs is chosen avoiding the testing of multiple inputs which cause identical output.

COMPLETENESS: the property that all necessary parts of the entity in question are included. Completeness of a product is often used to express the fact that all requirements have been met by the product.

CONSISTENCY: the property of logical coherency amoung constituent parts. Consistency may also be expressed as adherence to a given set of rules.

CORRECTNESS: the extent to which software is free from design and coding defects, i.e. fault free. It is also the extent to which software meets its specified requirements and user objectives. (IEEE Software Engineering Terminology)

DEBUGGING: the process of correcting syntactic and logical errors detected during coding. With the primary goal of obtaining an executing piece of code, debugging shares with testing certain techniques and strategies but differs in its usual ad hoc application and local scope.

DESIGN BASED FUNCTIONAL TESTING: the application of test data derived through functional analysis (see FUNCTIONAL TESTING) extended to include design functions as well as requirement functions.

DRIVER: code which sets up an environment and calls a module for test.

DYNAMIC ANALYSIS: involves execution or simulation of a development phase product. It detects errors by analyzing the response of a product to sets of input data.

EXTREMAL TEST DATA: test data that is at the extremes, or boundaries, of the domain of an input variable or which produces results at the boundaries of an output domain.

FORMAL ANALYSIS: uses rigorous mathematical techniques to analyze the algorithms of a solution. The algorithms may be analyzed for numerical properties, efficiency, and/or correctness.

FUNCTIONAL TESTING: application of test data derived from the specified functional requirements without regard to the final program structure.

INSPECTION: a manual analysis technique in which the program (requirements, design, or code) is examined in a very formal and disciplined manner to discover errors.

INSTRUMENTATION: the insertion of additional code into the program in order to collect information about program behavior during program execution.

INVALID INPUT (TEST DATA FOR INVALID INPUT DOMAIN): test data that lies outside the domain of the program's function.

PATH TESTING: a test method satisfying coverage criteria that each logical path through the program be tested. Often paths through the program are grouped into a finite set of classes; one path from each class is then tested.

PROOF OF CORRECTNESS: the use of techniques of mathematical logic to infer that a relation between program variables assumed true at program entry implies that another relation between program variables holds at program exit.

REGRESSION TESTING:Rerunning test cases which a program has previously executed correctly in order to detect errors created during software correction or modification activities.

SIMULATION: use of an executable model to represent the behavior of an object. During testing the computational hardware, the external environment, and even code segments may be simulated.

SPECIAL TEST DATA: test data based on input values that are likely to require special handling by the program.

STATEMENT TESTING: a test method satisfying the criterion that each statement in a program be executed at least once during program testing.

STATIC ANALYSIS: direct analysis of the form and structure of a product without executing the product. It may be applied to the requirements, design or code.

STRESS TESTING: see BOUNDARY VALUE ANALYSIS.

STUB: special code segments that when invoked by a code segment under test will simulate the behavior of designed and specified modules not yet constructed.

SYMBOLIC EXECUTION: an analysis technique that derives a symbolic expression for each program path.

TEST DATA SET : set of input elements used in the testing process.

TEST DRIVER: a program which directs the execution of another program against a collection of test data sets. Usually, the test driver records and organizes the output generated as the tests are run.

TEST HARNESS: see TEST DRIVER.

TESTING: examination of the behavior of a program by executing the program on sample data sets.

VALID INPUT (TEST DATA FOR A VALID INPUT DOMAIN): test data that lies within the domain of the function represented by the program.

VALIDATION: determination of the correctness of the final program or software produced from a development project with respect to the user needs and requirements.

VERIFICATION: in general, the demonstration of consistency, completeness, and correctness of the software at each stage and between each stage of the development lifecycle.

WALKTHROUGH: a manual analysis technique in which the module author describes the module's structure and logic to an audience of colleagues.

NOTE: Most of the definitions above are from:

ADRION,W.R.,BRANSTAD,M.A.,and CHERNIAVSKY,J.C.,"Validation, Verification, and Testing," NBS Special Publicaiton 500-75.

BIBLIOGRAPHY

[ADRI] ADRION, W.R., BRANSTAD, M.A., CHERNIAVSKY, J.C., "Validation, Verification, and Testing of Computer Software," NBS Special Publication 500-75, February 1981.

[BRAN] BRANSTAD, M.A., CHERNIAVSKY, J.C. and ADRION, W.R., "Validation, Verification and Testing for the Individual Programmer," NBS Special Publication 500-56,Washington,D.C., October 1979.

[CAIN] CAINE, S.H. and GORDON, E.K., "PDL: A Tool for Software Design," Proceedings of the National Computer Conference, 1975.

[DARR] DARRINGER, J.C. King, "Applications of Symbolic Execution to Program Testing," Computer, April 1978.

[DEMI] DEMILLO, R.A., LIPTON, R.J. and SAYWARD, F.C., "Hints on Test Data Selection: Help for the Practicing Programmer," Computer, April 1978.

[DUKE] DUKE, M.O., "Testing in a Complex Systems Environment," IBM Systems Journal, Vol. 14, No. 4, 1975.

[ELSP] ELSPAS, B.K., LEVITT, N. and WALDINGER, R.J. "An Assessment of the Techniques for Proving Program Correctness," ACM Computing Surveys, Vol. 4, No. 2, June 1972, pp. 97-147.

[ELME] ELMENDORF, W.R., "Disciplined Software Testing," Debugging Techniques in Large Systems, R. Rustin, ed., Prentice Hall, 1971, pp. 137-140.

[FAIR] FAIRLEY, R.E., "Static Analysis and Dynamic Testing of Computer Software," Computer, April 1978.

[FIPS38] "Guidelines for Documentation of Computer Programs and Automated Data Systems," NBS Federal Information Processing Standards Publication 38, February 1976.

[FIPS64] "Guidelines for Documentation of Computer Programs and Automated Data Systems for the Initiation Phase," NBS Federal Information Processing Standards Publication 64, August 1979.

[FOSD] FOSDICK, L.D. and OSTERWEIL, L.J. "Data Flow Analysis in Software Reliability," ACM Computing Surveys, Vol. 8, No. 3, September 1976, pp. 305-330.

[GERH] GERHART, S.L. and YELOWITZ, L. "Observations of Fallibility in the Application of Modern Programming Methodologies," IEEE Transactions on Software Engineering, Vol. SE-2, No. 3, September 1976, pp. 195-207.

[GLAS] GLASS, R., "Software Reliability Guidebook," Englewood Cliffs, NJ., Prentice-Hall, 1979.

[GOOD] GOODENOUGH, J.B. and GERHART, S.L. "Toward a Theory of Test Data Selection," *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 2, June 1975, pp. 156-173.

[HART] HARTWICK, R.D., "Test Planning," *AFIPS Conference Proceedings*, Vol. 46, 1977 NCC, pp. 285-294.

[HETZ] HETZEL, W.C., "Program Test Methods", Englewood Cliffs, NJ., Prentice-Hall, 1973.

[HOUGa] HOUGHTON, Raymond C., Jr., "Features of Software Development Tools," *NBS Special Publication 500-74*, Washington, D.C., February 1981.

[HOUGb] HOUGHTON, Raymond C. Jr., and OAKLEY, Karen A., eds., "NBS Software Tools Database," *NBSIR 80-2159*, October 1980.

[HOWDa] HOWDEN, W.E., "Reliability of the Path Analysis Testing Strategy," *IEEE Transactions on Software Engineering*, September 1976, pp. 208-214.

[HOWDb] HOWDEN, W.E., "Theoretical and Empirical Studies of Program Testing," *IEEE Transactions on Software Engineering*, Vol. SE-4, July 1976.

[HUAN] HUANG, J.C., "An Approach to Program Testing," *ACM Computer Surveys*, Vol. 7, No. 3, September 1975.

[MILLa] MILLER, E.F., Jr., "Program Testing: Art Meet Theory,"*Computer*, July 1977.

[MILLb] MILLER, E.F. and HOWDEN, W.E., "Tutorial: Software Testing and Validation Techniques," Long Beach, CA, *IEEE Computer Society*, 1978.

[MULL] MULLIN, F.J., "Software Test Management," *COMPSAC 77 Proceedings*, Chicago, Illinois, November 1977.

[MYER] MYERS, G.J., "The Art of Software Testing," New York, Wiley, 1979.

[PANZ] PANZL,D.J., "Automatic Software Test Drivers," *Computer*, April 1978.

[POWE] POWELL, P.B.,"Software Validation, Verification, and Testing - Technique and Tool Reference Guide," *NBS Special Publication 500-93*, Washington, D.C., September 1982.

[SAMM] "SAMM (Systematic Activity Modeling Method) Primer," BCS 10167, October 1978.

[TEIC] TEICHROEW, D. and HERSHEY,E., "PSA/PSL: A Computer-Aided Technique for Structured Documentation of Information Processing Systems," *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 1, 1977.

INDEX

Acceptance 6, 12, 54, 56
Acceptance criteria 8
Acceptance tests 51, 58
Algorithm 19, 25, 72
Assertion 17, 26

Bottom-up testing 18
Boundary value analysis 72
Branch testing 56, 72

Cause-effect graphing 17
Change request 13
Code V,V&T 25-26
Comparator 19
Completeness 5, 72
Complexity analysis 19
Consistency 5, 72
Correctness 5, 72
Critical design review 56

Design language 60
Design review 54
Design V,V&T 23
Desk checking 16
Detailed design specification 10
Development phase 6, 8
Dynamic analysis 16, 20-21, 23-24, 72
Dynamic assertion processor 18

Formal analysis 19-20, 25, 72

Initiation phase 7
Inspections 22
Installation plan 12
Installation report 12
Installation subphase 12
Instrumentation 18, 73
Integration testing 11

Module testing 57, 61

NBS software tools database 27

Operation and maintenance phase . . . 13

Planning 28-29

Preliminary design specification . . . 9
Problem reports 10, 12-13
Problem solving 1, 4, 28
Programming and testing subphase . . . 11
Project plan 8-9, 28

Regression testing 14, 69, 73
Requirements 8
Requirements based test cases 9
Requirements document 8
Requirements subphase 8
Requirements V,V&T 21

Software 1
Software development 1, 4
Software evaluation 13
Software test case specification . . . 8, 10-11
Specification based functional testing 17
Static analysis 15-16, 20-22, 25, 73
System testing 11

Test data preparation 17
Test execution 17-18
Top-down testing 18

V,V&T planning 28-29
Validation, verification, and testing 2, 4
Validation, verification, and testing plan 8

Walkthroughs 16, 22, 24

NBS-114A (REV. 2-80)

U.S. DEPT. OF COMM.

BIBLIOGRAPHIC DATA SHEET (See instructions)

1. PUBLICATION OR REPORT NO.
NBS SP 500-98

2. Performing Organ. Report No.

3. Publication Date
November 1982

4. TITLE AND SUBTITLE

Computer Science and Technology:
Planning for Software Validation, Verification, and Testing

5. AUTHOR(S)

Patricia B. Powell, Editor

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

Boeing Computer Services Co.
Seattle, Washington 98124

7. Contract/Grant No.
NB79SBCA0102

8. Type of Report & Period Covered
Final

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

National Bureau of Standards
Department of Commerce
Washington, DC 20234

10. SUPPLEMENTARY NOTES

Library of Congress Catalog Card Number: 82-600644

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

Today, providing computer software involves greater cost and risk than providing computer equipment. One major reason is hardware is mass-produced by proven technology, while software is still produced primarily by the craft of individual computer programmers. The document is for those who direct and those who implement computer projects; it explains the selection and use of validation, verification, and testing (V,V&T) tools and techniques for software development. A primary benefit of practicing V,V&T is increasing confidence in the quality of the software. The document explains how to develop a plan to meet specific software V,V&T goals.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

Key words: automated software tools; software lifecycle; software testing; software verification; test coverage; test data generation; validation.

13. AVAILABILITY

☒ Unlimited
☐ For Official Distribution. Do Not Release to NTIS
☒ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.
☐ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES
89

15. Price
$5.50

USCOMM-DC 6043-P80

79

☆U.S. GOVERNMENT PRINTING OFFICE: 1982-380-997/2327

Blank Page

ANNOUNCEMENT OF NEW PUBLICATIONS ON COMPUTER SCIENCE & TECHNOLOGY

Superintendent of Documents,
Government Printing Office,
Washington, DC 20402

Dear Sir:

Please add my name to the announcement list of new publications to be issued in the series: National Bureau of Standards Special Publication 500-.

Name ______________________________

Company ______________________________

Address ______________________________

City ______________ State ____________ Zip Code __________

(Notification key N-503)

NBS TECHNICAL PUBLICATIONS

PERIODICALS

JOURNAL OF RESEARCH—The Journal of Research of the National Bureau of Standards reports NBS research and development in those disciplines of the physical and engineering sciences in which the Bureau is active. These include physics, chemistry, engineering, mathematics, and computer sciences. Papers cover a broad range of subjects, with major emphasis on measurement methodology and the basic technology underlying standardization. Also included from time to time are survey articles on topics closely related to the Bureau's technical and scientific programs. As a special service to subscribers each issue contains complete citations to all recent Bureau publications in both NBS and non-NBS media. Issued six times a year. Annual subscription: domestic $18; foreign $22.50. Single copy, $4.25 domestic; $5.35 foreign.

NONPERIODICALS

Monographs—Major contributions to the technical literature on various subjects related to the Bureau's scientific and technical activities.

Handbooks—Recommended codes of engineering and industrial practice (including safety codes) developed in cooperation with interested industries, professional organizations, and regulatory bodies.

Special Publications—Include proceedings of conferences sponsored by NBS, NBS annual reports, and other special publications appropriate to this grouping such as wall charts, pocket cards, and bibliographies.

Applied Mathematics Series—Mathematical tables, manuals, and studies of special interest to physicists, engineers, chemists, biologists, mathematicians, computer programmers, and others engaged in scientific and technical work.

National Standard Reference Data Series—Provides quantitative data on the physical and chemical properties of materials, compiled from the world's literature and critically evaluated. Developed under a worldwide program coordinated by NBS under the authority of the National Standard Data Act (Public Law 90-396).

NOTE: The principal publication outlet for the foregoing data is the Journal of Physical and Chemical Reference Data (JPCRD) published quarterly for NBS by the American Chemical Society (ACS) and the American Institute of Physics (AIP). Subscriptions, reprints, and supplements available from ACS, 1155 Sixteenth St., NW, Washington, DC 20056.

Building Science Series—Disseminates technical information developed at the Bureau on building materials, components, systems, and whole structures. The series presents research results, test methods, and performance criteria related to the structural and environmental functions and the durability and safety characteristics of building elements and systems.

Technical Notes—Studies or reports which are complete in themselves but restrictive in their treatment of a subject. Analogous to monographs but not so comprehensive in scope or definitive in treatment of the subject area. Often serve as a vehicle for final reports of work performed at NBS under the sponsorship of other government agencies.

Voluntary Product Standards—Developed under procedures published by the Department of Commerce in Part 10, Title 15, of the Code of Federal Regulations. The standards establish nationally recognized requirements for products, and provide all concerned interests with a basis for common understanding of the characteristics of the products. NBS administers this program as a supplement to the activities of the private sector standardizing organizations.

Consumer Information Series—Practical information, based on NBS research and experience, covering areas of interest to the consumer. Easily understandable language and illustrations provide useful background knowledge for shopping in today's technological marketplace.

Order the **above** *NBS publications from: Superintendent of Documents, Government Printing Office, Washington, DC 20402.*

Order the **following** *NBS publications—FIPS and NBSIR's—from the National Technical Information Services, Springfield, VA 22161.*

Federal Information Processing Standards Publications (FIPS PUB)—Publications in this series collectively constitute the Federal Information Processing Standards Register. The Register serves as the official source of information in the Federal Government regarding standards issued by NBS pursuant to the Federal Property and Administrative Services Act of 1949 as amended, Public Law 89-306 (79 Stat. 1127), and as implemented by Executive Order 11717 (38 FR 12315, dated May 11, 1973) and Part 6 of Title 15 CFR (Code of Federal Regulations).

NBS Interagency Reports (NBSIR)—A special series of interim or final reports on work performed by NBS for outside sponsors (both government and non-government). In general, initial distribution is handled by the sponsor; public distribution is by the National Technical Information Services, Springfield, VA 22161, in paper copy or microfiche form.

U.S. Department of Commerce
National Bureau of Standards

Washington, D.C. 20234
Official Business
Penalty for Private Use $300

U.S.MAIL

POSTAGE AND FEES PAID
U.S. DEPARTMENT OF COMMERCE
COM-215

THIRD CLASS
BULK RATE